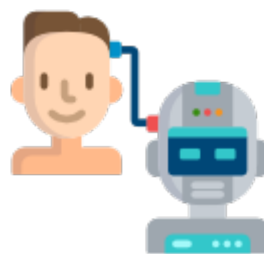
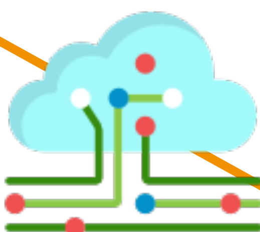
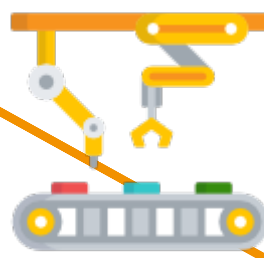
S V E U Č I L I Š T E U S P L I T U
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE U SPLITU





Uvod u umjetnu inteligenciju

Darko Stipaničev - Ljiljana Šerić - Maja Braović

Uvod u umjetnu inteligenciju

UDŽBENICI SVEUČILIŠTA U SPLITU
MANUALIA UNIVERSITATIS STUDIORUM SPALATENSIS

UVOD U UMJETNU INTELIGENCIJU

Darko Stipaničev – Ljiljana Šerić – Maja Braović



Split, 2021.

UDŽBENICI SVEUČILIŠTA U SPLITU
MANUALIA UNIVERSITATIS STUDIORUM SPALATENSIS

Nakladnik



FAKULTET ELEKTROTEHNIKE,
STROJARSTVA I BRODOGRADNJE U SPLITU

Urednik

prof. dr. sc. Vladan Papić

Autori

prof. dr. sc. Darko Stipaničev
izv. prof. dr. sc. Ljiljana Šerić
doc. dr. sc. Maja Braović

Recenzenti

prof. dr. sc. Tamara Grujić
izv. prof. dr. sc. Saša Mladenović
izv.prof.dr.sc. Damir Krstinić
doc. dr. sc. Toni Jakovljević

Lektor

Mišo Sučević, prof.

Grafička priprema i autor fotografija u uvodnim dijelovima poglavlja

prof. dr. sc. Darko Stipaničev



Manualia universitatis studiorum Spalatiensis
(Udžbenici Sveučilišta u Splitu)

Odobreno odlukom Senata Sveučilišta u Splitu na 39. sjednici održanoj 17.prosinca 2020.g.
klasa: 602-04/20-01/00039; ur.broj: 2181-202-03-/7-20-0015

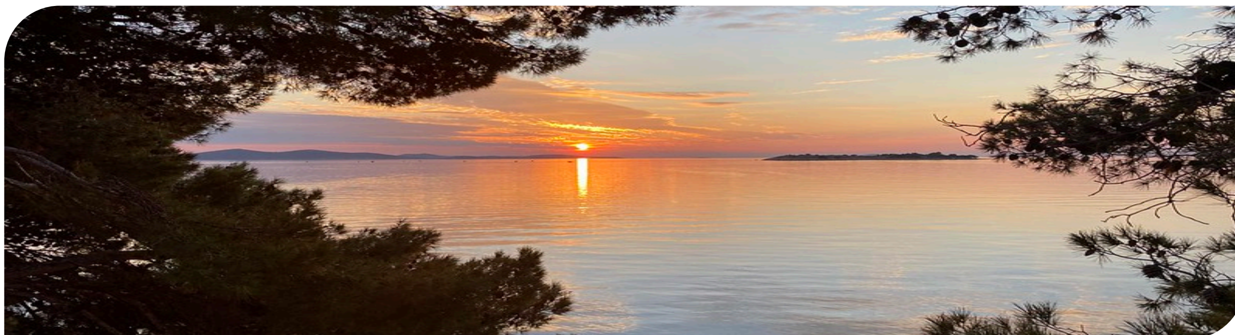
ISBN 978-953-290-108-5



Prvo izdanje objavljeno u studenom 2021.g.

SADRŽAJ

PREDGOVOR



”

Ovim smo udžbenikom čitateljima pokušali približiti i objasniti osnovne pojmove vezane uz nebiološku inteligenciju, koja osim povijesno najstarije „dobre staromodna umjetne inteligencije“ obuhvaća i računsku i distribuiranu inteligenciju, te pokazati da se ponekad za rješavanje kompleksnih problema mogu primijeniti relativno jednostavna programska rješenja.

Područje računalnih znanosti koje se bavi inteligentnim tehnologijama danas je industrija koja zapošljava velik broj stručnjaka, pa je kolegij koji uvodi studente u tehničke primjene inteligentnih tehnologija sastavni dio svih studijskih programa računalnih znanosti. Tako je i u studijskim programima računarstva i računalnog inženjerstva na FESB-u gdje je kolegij „*Umjetna inteligencija*” obavezni kolegij, a na ostalim studijima izborni. Naziv kolegija trebao bi biti *„Uvod u umjetnu inteligenciju”* s obzirom na to da je ovo područje danas toliko veliko da se u okviru kolegija tjednog opterećenja 2 sata predavanja i 2 sata vježbi, može stvarno dati samo uvod u umjetnu inteligenciju.

Ovim smo udžbenikom čitateljima pokušali približiti i objasniti osnovne pojmove vezane uz nebiološku inteligenciju, koja osim povijesno najstarije „*dobre staromodna umjetne inteligencije*” (engl. *GOFAI – Good Old Fashioned Artificial Intelligence*) obuhvaća i računsku i distribuiranu inteligenciju, te pokazati da se ponekad za rješavanje ***kompleksnih problema*** (engl. *Complex Problems*)[1] kriju relativno jednostavna programska rješenja. Udžbenik je prije svega posvećen *„dobroj staromodnoj umjetnoj inteligenciji”* koja nam daje temeljna znanja ovog područja. Bavimo se problemom rješavanja kompleksnih zadataka koje nije moguće riješiti klasičnim algoritamskim načinom, već je za njihovo rješenje potrebno specifično znanje koje koristi čovjek prilikom njihovog rješavanja. Zbog toga je pojam ***znanja*** specifičan za sve inteligentne tehnologije. Kod „*dobre staromodne umjetne inteligencije”* znanje je prisutno cijelo vrijeme prilikom funkcioniranja sustava i pohranjeno je u posebne strukture koje se nazivaju ***baze znanja***, a kod računske inteligencije znanje je prisutno na početku kada se projektira sustav računske inteligencije. Upravo zbog toga centralni dio ovog udžbenika posvećen je znanju, njegovom prikupljanju, kodiranju i pohrani kao i njegovoj primjeni kod zaključivanja i rasuđivanja.

Drugo važno područje koje ovaj udžbenik obrađuje su specifične metode rješavanja kompleksnih zadataka koje nazivamo ***pretraživanje***. Kompleksne zadatke najčešće rješavamo pretražujući bazu

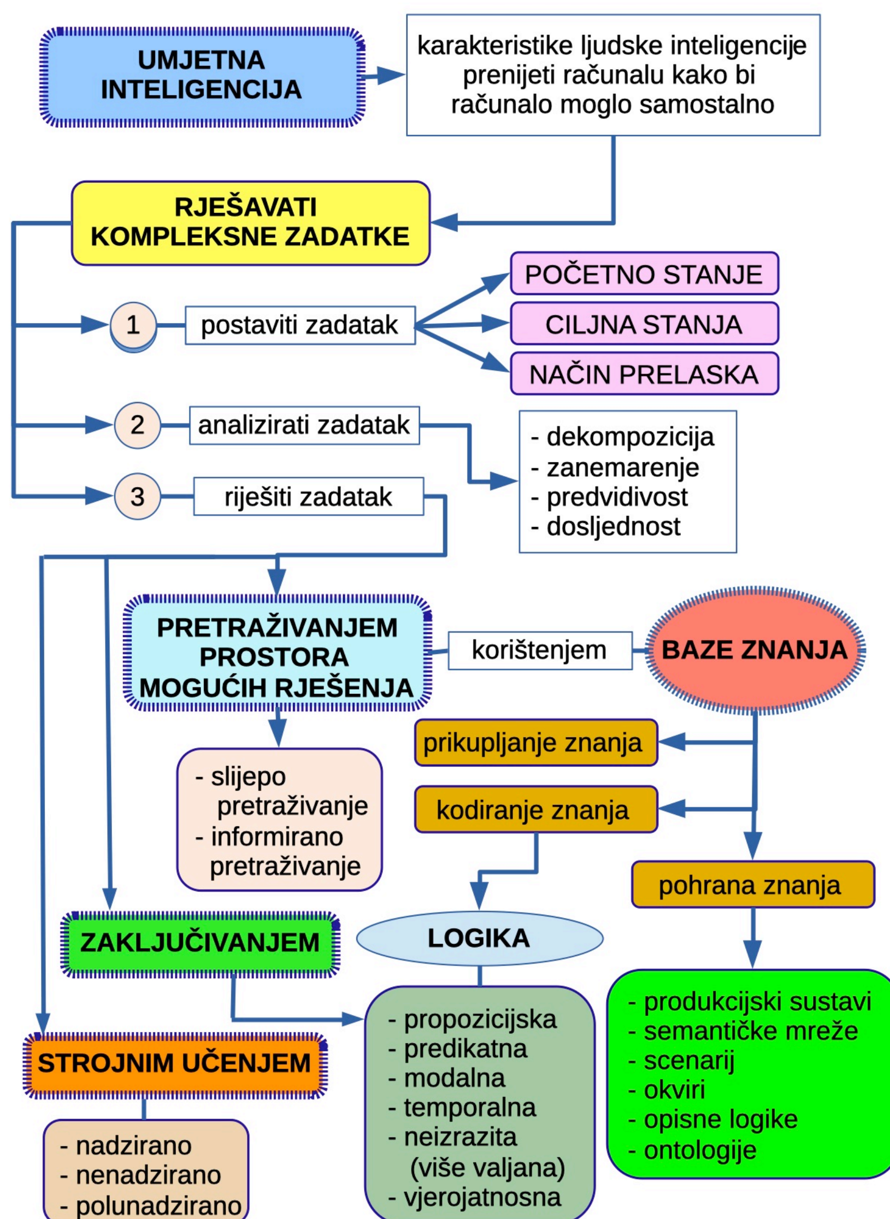
[1] Engleska riječ „*complex”* na hrvatski se najčešće prevodi kao „*složen”* iako bi joj puno više odgovarao prijevod „*kompleksan”,* kojem smo se i mi u ovom tekstu priklonili. Složeno je ono što se sastoji od više različitih dijelova, a kompleksno uz to može biti i zamršeno, zapleteno i komplicirano, a takvi su upravo zadaci kojima se bavi umjetna inteligencija.

znanja kako bismo sustavno gradili prostor rješenja zadatka i pronašli put od početnih stanja do nekog od ciljnih stanja koja nam daju rješenje kompleksnog zadatka.

Treće važno područje kojim se umjetna inteligencija bavi je područje učenja i to posebno ***strojnog učenja***, pa je jedan dio ovog udžbenika posvećen upravo toj temi.

Kako je umjetna inteligencija dio računalnih znanosti, ne možemo ništa bez programiranja i programskih jezika. U okviru umjetne inteligencije razvili su se specifični programski jezici ***Lisp*** i ***Prolog*** kojima je posvećen dio ovog udžbenika i jedan dio laboratorijskih vježbi kolegija. Treći jezik koji u posljednje vrijeme postaje značajan u umjetnoj inteligenciji je ***Python***, pa je i on našao mjesto u ovom udžbeniku.

Kako se kretati kroz poglavlja, najbolje ilustrira slika 1-1 koja daje i kratki pregled svega što ovaj udžbenik obuhvaća.



Slika 1-1. Teme iz područja umjetne inteligencije koje obrađuje ovaj udžbenik

Nadamo se da će nekog od čitatelja udžbenik potaknuti na samostalna istraživanja i produbljivanja znanja iz drugih područja. Zbog toga smo, gdje je god bilo moguće, ukazali na literaturu kojom se znanje može produbiti. Posebnu smo pažnju kod pisanja teksta posvetili i ljudima koji su doprinijeli razvoju ovog područja, pa na puno mjesta uz spomen imena dajemo i godine kada je autor živio i stvarao, a čitatelje pozivamo da zavire i u njihove biografije. Znanost je veliki mozaik koji slaže puno, puno znanstvenika bez kojih ne bi bilo ni znanstvenih doprinosa. Poznavanje engleskih naziva pojmova u tehnici je vrlo važno, pa smo uza sve značajne pojmove u zagradama dali u kurzivu i njihove engleske nazive i kratice.

Iza svakog poglavlja u odvojenom dijelu je mali dodatak koji je posvećen zanimljivim primjerima primjene umjetne inteligencije i inteligentnih tehnologija u različitim područjima ljudske djelatnosti.

Udžbenik prati i web-portal na adresi http://ai.fesb.hr koji smo zamislili kao središnje mjesto na kojem ćemo postaviti digitalnu verziju udžbenika, te ga povremeno nadopunjavati sadržajima koji nisu stali u sam udžbenik. Zbog toga pozivamo čitatelje da povremeno posjete ovaj web-portal.

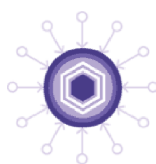
Napomena: Sve ilustracije originalno su rađene za ovaj udžbenik, a sve ikonice koje su korištene u njihovoj izradi su sa slobodnom, ne-ekskluzivnom, besplatnom, prenosivom licencom (engl. *Non-exclusive, Royalty-free, Transferable, Worldwide Licence)* preuzete s web-stranice https://ya-webdesign.com. Sve fotografija su fotografije autora ili fotografije sa slobodnom licencom, na primjer s web-stranice https://pixabay.com, dok su kod nekih fotografija licencna prava spomenuta uz njih.

....









Dodatak: Jedan od najdugoročniji projekata umjetne inteligencije je projekt ***Cyc*** (čita se *Sajk*) kojeg je 1984.g. pokrenuo ***Douglas Lenat***. Projekt je aktivan još i danas u okviri tvrtke ***Cycorp***[2] kojoj je na čelu Lenat. Cilj projekta je bio (i ostao) stvoriti sveobuhvatnu ontologiju i bazu znanja koja bi obuhvaćala sve ono što mi znamo u okviru našeg ***zdravorazumskog znanja*** (engl. *Common Sense Knowledge*), te je kodirati u strojno razumljivom obliku, korištenjem posebno razvijenog ontologijskog jezika ***CycL***. Cyclova ontologija danas uključuje par miliona termina podijeljenih u individualne pojmove (entitete), predikate (relacije, atribute, svojstva, funkcije) i kolekcije. Primjeri su: `(#$capitalCity #$France #$Paris)` koji formalno iskazuje tvrdnju *„Pariz je glavni grad Francuske."* ili `(#$isa #$BillClinton #$UnitedStatesPresident)` koja znači *„Bill Clinton pripada kolekciji Predsjednici SAD-a"*.

Cyc ima i ***mehanizam zaključivanja*** (engl. *Inference Engine*) kojim se pokušaju izvući odgovori iz baze znanja na postavljeni upit. Zaključuje na temelju različitih postupaka logičkog zaključivanja o kojima detaljnije govorimo u poglavlju 5. Iako je imao dosta kritičara Cyc je na neki način prethodnik projekta ***IBM Watson***[3], računalnog sustava namijenjenog ***davanju odgovora na postavljena pitanja*** (engl. *QA – Question Answering System*) zadana prirodnim jezikom. IBM je na Watsonu počeo raditi 2005.g. da bi 2011.g. Watson nastupio i pobijedio u kvizu ***Jeopardy!*** u kojem natjecatelji nastoje pogoditi pravo pitanje na prikazani odgovor vezan uz ***opće znanje*** (engl. *Global Knowledge*). Primjer je odgovor, za koji Watson nije postavio točno pitanje u kategoriji *„Gradovi SAD-a": „Njegov najveći aerodrom ima ime po junaku 2. svjetskog rata, a drugi iza njega po bitci 2. svjetskog rata."* Točan odgovor je *„Što je Chicago?"*, a Watson je pogriješio postavivši pitanje *„Što je Toronto?"*. Ipak je tijekom cijelog natjecanja Watson bio uvjerljivo bolji od ostala dva natjecatelja. Danas nalazi primjenu u brojnim područjima primjerice u zdravstvenoj zaštiti, edukaciji, kulinarstvu, financijama, oglašavanju, itd. Baza znanja Watsona se sastoji i od strukturiranih i od nestrukturiranih podataka.

[2] Više detalja na https://cyc.com.

[3] Ime ***Watson*** je dobio u spomen na osnivača IBM-a ***Thomasa Johna Watsona*** (1874.–1956.). Više detalja na https://www.ibm.com/watson.

1 INTELIGENCIJA I UMJETNA INTELIGENCIJA





Što je umjetna inteligencija? Definicija ima puno, a možda je najjednostavnija od njih ona koja umjetnu inteligenciju definira kao „znanost kojoj je cilj napraviti stroj – računalo sposobno obavljati postupke koje u ovom trenutku čovjek obavlja bolje“. Umjetna inteligencija spada u područje nebiološke inteligencije ili alfa–inteligencije koja osim umjetne inteligencije uključuje i računsku inteligenciju i distribuiranu inteligenciju.

Još od antičkog doba ljudi pokušavaju odgovoriti na pitanje što se događa u ljudskom mozgu. Kako čovjek misli, prepoznaje objekte, donosi zaključke i odluke te povratno djeluje na svoju okolinu? Što karakterizira taj dio njegovog ponašanja koji možemo nazivati inteligentno ponašanje? Što je u biti inteligencija? Je li ona svojstvena samo čovjeku? Pitanja je puno, a odgovora još i više. Filozofi, psiholozi, sociolozi, neurolozi, neuropsiholozi, svaki sa svog stajališta pokušavaju dati odgovore na ova pitanja. S druge strane, još od antičkog doba postojala je i druga struja koju su činili praktičari, znanstvenici i inženjeri područja koje danas nazivamo tehničke znanosti. Oni se nisu toliko opterećivali pitanjima vezanim uz prirodu ***biološke inteligencije*** (engl. *BI - Biological Intelligence*), već ih je prije svega zanimalo mogu li se te značajke biološke inteligencije prenijeti stroju. Može li se napraviti stroj koji će pokazivati svojstva biološke inteligencije? Ako može, takvu bi inteligenciju trebali zvati nebiološka inteligencija (engl. *N-BI – Non-Biological Intelligence*).

Ova su istraživanja posebno potaknuta razvojem računala, tako da se krajem pedesetih godina za ovaj tip istraživanja uvodi i formalni naziv ***umjetna inteligencija*** (engl. *AI - Artificial Intelligence*) sa željom da on obuhvati sva istraživanja vezana uz problematiku prenošenja svojstava biološke inteligencije stroju, prije svega računalu, s ciljem da tada takvo računalo bude u mogućnosti:

- rješavati probleme, kompleksne zadatke koje najčešće nazivamo problemski zadaci koje klasični programi ne mogu riješiti ili mogu riješiti, ali uz velike napore (dugo vrijeme trajanja izvođenja programa, te veliki troškovi za pisanje i implementiranje programa).

Ovo je bio jedan od ciljeva projekta japanske pete generacije računala nazvanog ***„Fifth Generation Computer Systems [Present and Future] (FGCS)”*** *(*hrv. *peta generacija računalnih sustava).* Projekt najavljen u listopadu 1981. godine predviđao je temeljitu promjenu računarstva u drugoj polovini 90-ih godina 20. stoljeća. Ideja je bila napraviti računalo koje s korisnikom komunicira prirodnim govorom i slikovnim prikazom, a uz to ima i sposobnost učenja, zaključivanja i odlučivanja.

Iako se to ni do danas nije ostvarilo, ipak su dostignuća u ovom području znanosti i tehnologije toliko odmakla da je nužno u postupku obrazovanja u području tehničkih znanosti steći osnovna znanja iz teorije na kojoj se temelje ovakvi sustavi.

Vratimo li se malo u prošlost, zanimljivo je uočiti da je u vrijeme pojave računala (ranih pedesetih godina 20. stoljeća) bilo nezamislivo da će računala ikada biti sposobna računati nešto više od balističke putanje projektila. Čak je i ***John Von Neumann*** (1903. – 1957.), otac računarstva, izjavio da računala nikada neće dosegnuti ni jednu osobinu inteligencije.

Međutim, već krajem 1950-ih i početkom 1960-ih godina javljaju se prvi počeci nečega iz čega se nešto kasnije razvila umjetna inteligencija. Radilo se o prvim pokušajima pisanja programa za igranje šaha. Upravo je zbog toga šah bio dugo vremena školski primjer na kojem su se provjeravale metode i postupci umjetne inteligencije. Danas, gotovo 60 godina kasnije, postoje programi koji pobjeđuju velemajstore, ali se za njih može kazati da ilustriraju samo neke aspekte inteligentnog ponašanja. Dobar igrač šaha je samo (ne više, ali ni manje) dobar igrač šaha, pa se područje umjetne inteligencije okrenulo širim područjima koje obuhvaća sam pojam inteligencija.

1.1 Što je biološka inteligencija?

Što je to inteligencija? Koje su značajke inteligencije? Formalna definicija potpisana od strane 52 znanstvenika udružena 1994. godine u grupu ***„Mainstream Science of Intelligence*** *(*hrv. *Temeljna znanost o inteligenciji)* objavljena 13. prosinca 1994. godine u časopisu *„Wall Street Journal”* sažeto kaže:

„Inteligencija je vrlo općenita mentalna sposobnost koja, između ostalog, uključuje sposobnost zaključivanja, planiranja, rješavanja problema, apstraktno razmišljanje, razumijevanje kompleksnih ideja, brzo učenje i učenje na temelju iskustva. Inteligencija ne obuhvaća samo učenje iz knjiga, usku akademsku vještinu ili elegantno rješavanje testova. Prije od toga ona reflektira širu i dublju sposobnost razumijevanja našeg okruženja – opažanja, shvaćanja smisla u stvarima ili odlučivanja o tome što napraviti.”

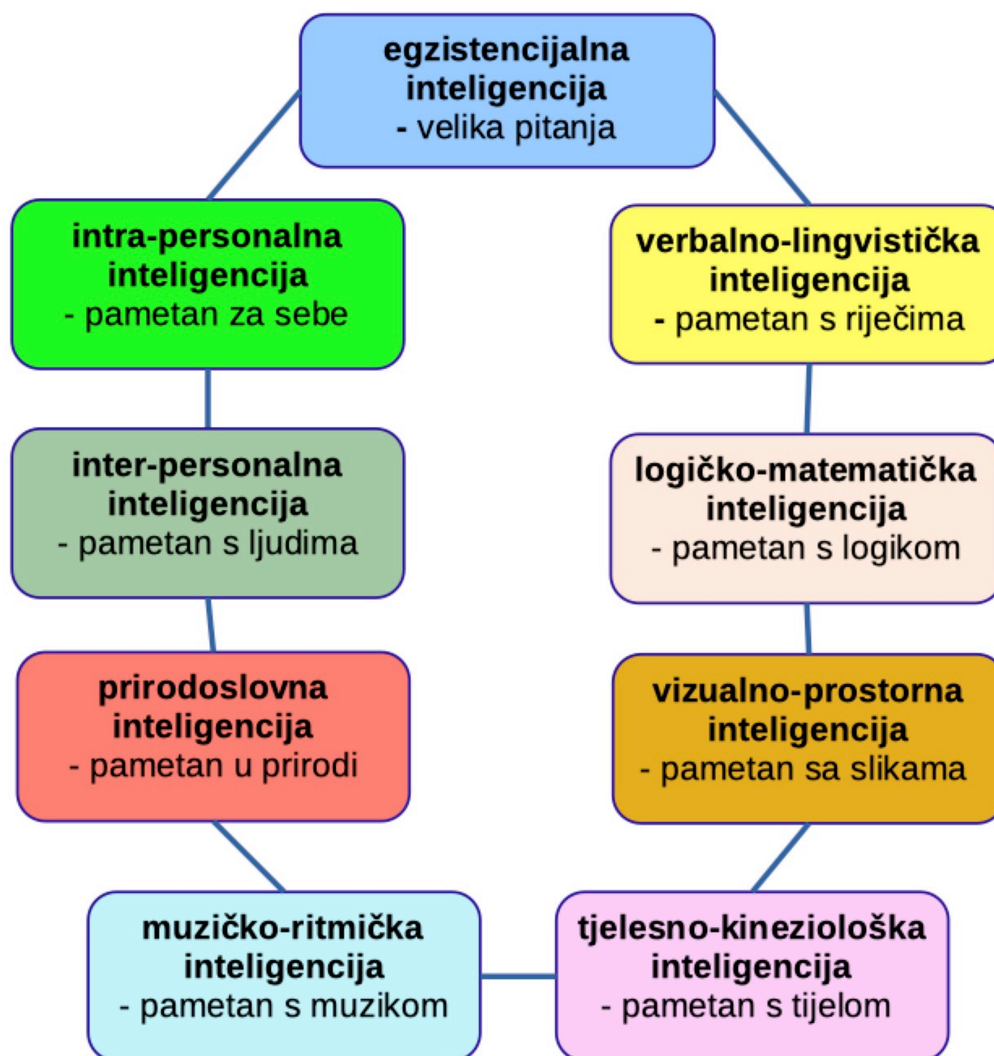
Godine 1995. grupa od 11 istraživača imenovanih od strane Američke udruge psihologa u svom izvještaju ***„Intelligence: Knowns and Unknowns”*** *(*hrv. *Inteligencija: poznato i nepoznato)* spomenula je da je dvadesetak eminentnih teoretičara i istraživača na pitanje definiranja inteligencije dalo dvadesetak različitih odgovora.

Iako postoje razilaženja u preciznom definiranju inteligencije, općenito je prihvaćen pristup nazvan ***višestruka inteligencija*** (engl. *multiple intelligence*) koji smatra da postoje različiti tipovi inteligencije. Čak i istraživači vezani uz dominantni faktor inteligencije nazvan g-faktor ili generalni faktor, ne odbacuju tezu o višestrukim inteligencijama navodeći da g-faktor može biti matematička (statistička) kombinacija različitih tipova inteligencije. Teoriju g-faktora ili generalne inteligencije postavio je 1904. godine engleski psiholog ***Charles Edward Spearman*** (1863. – 1945.) vjerujući da ljudi koji u jednom području pokažu dobre mentalne rezultate, iste će vjerojatno ponoviti i u drugom području.

Uz pojam višestruke inteligencije u medijima se često spominju pojmovi kao što su motorička inteligencija, emotivna inteligencije, socijalna inteligencija, logičko-matematička inteligencija pa čak i seksualna inteligencija. Ovim se pojmovima nastoje istaknuti pojedini tipovi ljudske inteligencije. I naravno, u jednoj jedinki ne trebaju biti prisutni baš svi. Vrsni sportaš, na primjer tenisač, sigurno ima vrlo razvijenu motoričku inteligenciju, a u isto vrijeme ne mora imati baš jako razvijenu logičko-matematičku inteligenciju. Višestruke inteligencije su posebno zanimljive za istraživanja vezana uz umjetnu inteligenciju pa se na njih detaljnije osvrćemo.

Dva su psihologa postavila formalne teorije višestrukih inteligencija. To su ***Howard Gardner*** (1943. –) sa Sveučilišta Harvard, koji je 1983. godine postavio teoriju višestrukih inteligencija, i ***Robert Sternberg*** (1922. – 2014.) koji je 1985. godine na Sveučilištu Tufts postavio trierarhijsku teoriju inteligencije.

Gardnerova teorija višestruke inteligencije nastala je na temelju psiholoških istraživanja razvoja inteligencije kod djece, nadarenih pojedinaca, ali i osoba povrijeđenog mozga. Najprije je 1985. godine predložio sedam osnovnih tipova inteligencije koje je kasnije proširio za još dva dodatna. Osnovni tipovi inteligencije po Gardneru su (slika 1-2):



Slika 1-2. Osnovni tipovi višestruke inteligencije prema Gardneru

1. ***Verbalno-lingvistička inteligencija*** – vezana uz napisane ili izgovorene riječi. Iskazuje sposobnost objašnjavanja, govora, podučavanja.

2. ***Logičko-matematička inteligencija*** – vezana uz logiku, apstrakciju, induktivno i deduktivno zaključivanje i baratanje brojevima.

3. ***Vizualno-prostorna inteligencija*** – vezana uz osjet vida, orijentaciju, prosuđivanja i apstrakciju prostora.

4. ***Tjelesno-kineziološka inteligencija*** – vezana uz pokret i rad. Ovo bi trebala biti prije spomenuta motorička inteligencija.

5. ***Muzičko-ritmička inteligencija*** – vezana uz ritam, glazbu i osjet sluha.

6. ***Prirodoslovna inteligencija*** – vezana uz prirodu, prehranu i klasifikaciju objekata u našem okružju. Jedna je od dvije naknadno dodane kategorije, ali je i dan danas kritičari osporavaju smatrajući da više iskazuje interese nego samu inteligenciju.

7. ***Interpersonalna inteligencija*** – vezana uz interakcije s ostalim ljudima i komunikaciju s njima, a iskazuje sposobnost empatije i prepoznavanja tuđih osjećaja. Za ljude kod kojih prevladava kažemo da su ekstrovertirani.

8. ***Intrapersonalna inteligencija*** – vezana uz nas same. Ljudi kod kojih prevladava ovaj tip inteligencije su introvertirani, okrenuti sebi, svjesni su sebe, svojih stanja i emocija, ali i ciljeva i motivacija.

9. ***Egzistencijalna inteligencija*** – ili kako je Gardner naziva *„inteligencija velikih pitanja”*, također je jedna od naknadno dodanih inteligencija vezana uz filozofska pitanja poput pitanja pojma vremena, života, smrti. To su velika pitanja o kojima malo ljudi može razmišljati, a još manje raspravljati.

Testovi inteligencije kojima se mjeri ljudski ***IQ*** (engl. *Intelligence Quotient*) uglavnom se odnose na logičko-matematičku, lingvističku i prostornu inteligenciju, a kako ćemo vidjeti i u ovom udžbeniku to su i područja kojima se uglavnom bavi umjetna inteligencija. Sve ove tipove inteligencije možemo grubo podijeliti u tri grupe: inteligencije vezane uz objekte (prostorna, logičko-matematička, tjelesno-kineziološka i prirodoslovna inteligencija), inteligencije koje nisu direktno vezane uz objekte (lingvistička i muzička inteligencija) i inteligencije koje su vezane uz kulturu i kulturološka shvaćanja (interpersonalna, intrapersonalna i egzistencijalna inteligencija).

Zanimljivo je kako se u kontekstu Gardnerove teorije može promatrati razvoj društva koji se može grubo podijeliti u poljoprivredno (agrarno), industrijsko i informacijsko društvo. U ***poljoprivrednom društvu*** čovjek je koristio pretežno prirodoslovnu inteligenciju. Bio je usredotočen na proizvodnju hrane koju su proizvodili radnici poljoprivrednici, koristeći vještinu proizašlu iz tjelesno-kineziološke inteligencije. Pojavom poljoprivredne mehanizacije znatno je smanjena potreba za ovakvim tipom rada, pa je društvo polako, preko određenih prijelaznih faza prešlo u ***industrijsko društvo***. Industrijsko se društvo temeljilo na prostornoj inteligenciji i bilo je koncentrirano na proizvodnju fizičkih umjetnih tvorevina (artefakta) koje su proizvodili radnici koristeći svoju tjelesno-kineziološku inteligenciju. Pojavom industrijskih strojeva i automatizacije, znatno je smanjena potreba za ovakvom vrstom rada. Sljedeća faza u kojoj se sada nalazimo je ***informacijsko društvo*** koje se temelji na logičko-matematičkoj inteligenciji. Koncentrirano je na proizvodnju reprezentacijskih umjetnih tvorevina koje stvaraju činovnici i uredski radnici koristeći svoju logičko-matematičku i verbalno-lingvističku inteligenciju. Reprezentacijski artefakti informacijskog društva prodaju se kao proizvod (razne komunikacijske usluge, produkti industrije zabave, usluge koje se nude na internetu…) ili služe u proizvodnji različitih fizičkih proizvoda (tehnologije, recepture, postupci proizvodnje…). U oba slučaja, kako bi se monopoliziralo tržište, važna je zaštita intelektualnog vlasništva pa je zato u industrijskom i informacijskom društvu poseban značaj patentne zaštite i *copyrighta*. Kao što su poljoprivredni i industrijski radnici postupno zamjenjivani efikasnijim strojevima, pa na primjer jedan traktor napravi više posla u jednom danu nego što je moglo napraviti stotine poljoprivrednih radnika, za očekivati je da će se i rad radnika informacijskog društva postupno zamjenjivati efikasnijim sustavima obrade informacija. Toga smo svakodnevno i svjedoci, spomenimo samo bar-kodni zapis koji se danas nalazi na svakom proizvodu i koji je zamijenio rad tisuća radnika u sektoru prodaje.

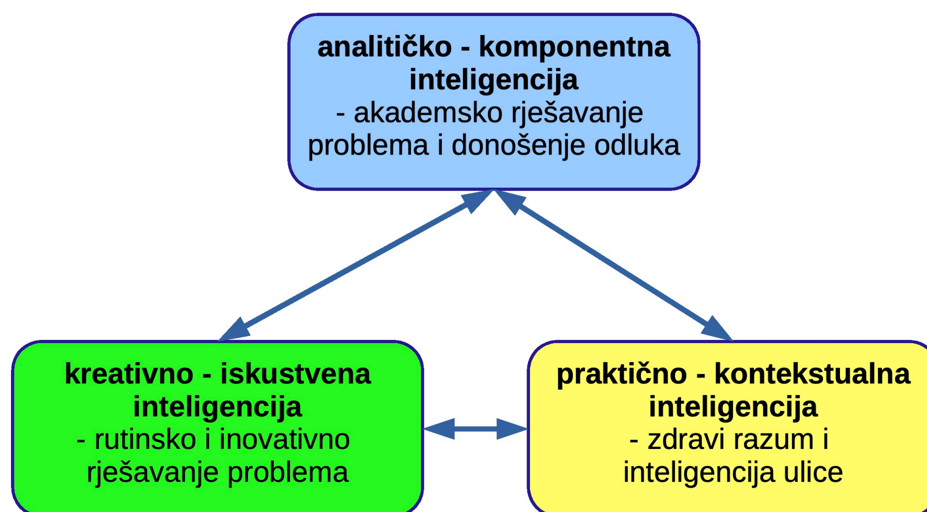
Međutim, zamjena odgovarajućeg rada efikasnijim napravama nije i potpuna zamjena čovjeka. Traktor još uvijek vozi čovjek, automatiziranim industrijskim strojevima upravljaju ljudi. Iako je ovim strojevima zamijenjen rad velikog broja ljudi, za njih ne možemo reći da su inteligentni. Efikasnost i učinkovitost rada, pa čak ako rade i potpuno samostalno, ne znači da su strojevi inteligentni, barem ne u širokom smislu riječi inteligencija. Oni u pojedinim dijelovima mogu pokazivati pojedine značajke inteligencije koje možemo svrstati u neke od tipova – prije svega logičko-matematičke, verbalno-lingvističke i vizualno-prostorne inteligencije.

Steinbergova ***trierarhijska teorija inteligencije*** (engl. *Triarhic Theory of Intelligence*) razlikuje tri osnovna tipa inteligencije: analitičku, kreativnu i praktičnu (slika 1-3) i vezana je prije svega uz način kako se čovjek prilagođava promjenama u svom okružju. Steinberg kaže da je inteligencija *„...mentalna aktivnost usmjerena prema namjernim, svrsishodnim promjenama, odabiru i oblikovanju realnog okružja bitnog za život jedinke...”*.

1. ***Analitičko-komponentna inteligencija*** (engl. *Analitic-Componential Intelligence*) uključuje procese koji se koriste kod rješavanja problema (kompleksnih problemskih zadataka) i donošenja odluka. Steinberg ove procese naziva ***meta-komponente***. Drugi dio su ***komponente djelovanja***, procesi koji provode akcije koje meta-komponente kreiraju. Uključuju sve ono na temelju čega mi djelujemo kao što je uočavanje problema, uočavanje odnosa između objekata i preslikavanje ovih relacija na druge skupove pojmova. Treći dio analitičke inteligencije su ***komponente prikupljanja znanja*** koje koristimo kada prikupljamo nove informacije. Uključuju selektiranje bitnih od nebitnih informacija, ali i kombiniranje informacija dobivenih različitim osjetilima.

2. ***Kreativno-iskustvena (empirijska) inteligencija*** (engl. *Creative-Experimental Intelligence*) vezana je uz vještinu provođenja određenih zadataka u odnosu na stupanj njihovog poznavanje. Razlikuje se ***nova situacija*** koju pojedinac nikada do sada nije iskusio i situacija za koju možemo kazati da u njenom rješavanju koristimo ***automatizaciju***. To znači da smo je do sada ponovili puno puta, pa je možemo izvesti bez puno razmišljanja (na primjer serviranje u tenisu). Čovjek koji posjeduje određenu automatiziranu vještinu ne mora biti uspješan kod snalaženja u novim situacijama.

3. ***Praktično-kontekstualna inteligencija*** (engl. *Practical-Contextual Intelligence*) je najuže vezana uz ljudsko okružje i sposobnost prilagodbe tom okružju. Uključuje procese prilagodbe, promjene i odabira, a cilj joj je uklopiti jedinku na najbolji mogući način u okružje. ***Prilagodba*** je vezana uz vlastitu promjenu kako bi se što bolje uklopili u okružje, ***promjena*** uključuje aktivnosti kojima mijenjamo okružje da bi nam ono što bolje odgovaralo, a ***odabir*** je biranje između novih mogućnosti kojima zamjenjujemo postojeće stanje kako bi što bolje ispunili vlastite ciljeve. Jedinka se može odlično uklapati u svoje okružje a da pri tome ne posjeduje značajnu analitičku inteligenciju.



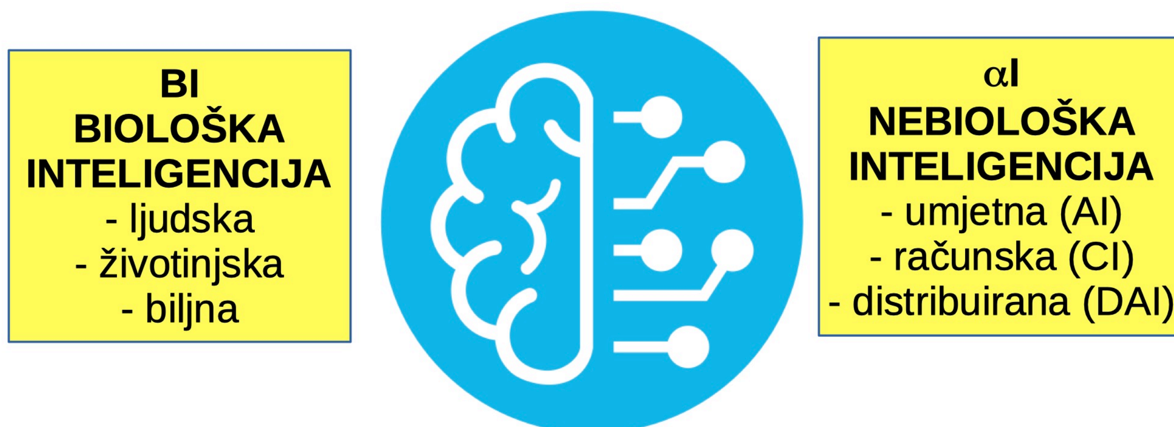
Slika 1-3. Osnovni tipovi višestruke inteligencije prema Steinbergu

Gardnerove i Steinbergove teorije višestrukih inteligencija međusobno se ne isključuju, već nadopunjuju.

1.2 Što je nebiološka inteligencija?

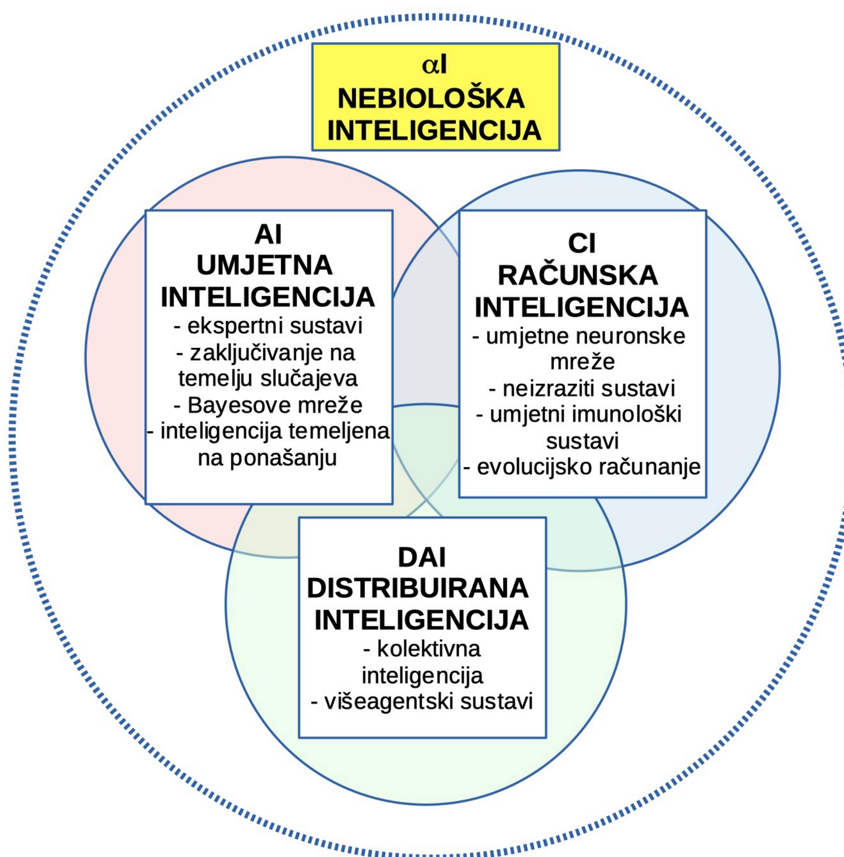
Umjetna inteligencija spada u ***nebiološku inteligenciju*** koja se ponekad naziva i ***αI – alfa inteligencija ili apstraktna inteligencija*** (engl. *Abstract Intelligence*), no to nije jedina nebiološka inteligencija. Uz nju se kao nebiološke inteligencije spominju i ***računska inteligencija*** (engl. *CI - Computational Intelligence*) i ***raspodijeljena ili distribuirana umjetna inteligencija*** (engl. *DAI -*

Distributed Artificial Intelligence). Zajedno s ***biološkom inteligencijom***, inteligencijom svojstvenom živim bićima, ova tri tipa inteligencije čine tzv. ***A B C D*** inteligencije nastale prema engleskim riječima ***A**rtificial* - ***B**iological* - ***C**omputational* - ***D**istributed Artificial Intelligence.*



Slika 1-4. Biološka i nebiološka inteligencija

U ovom se udžbeniku prvenstveno bavimo umjetnom inteligencijom, ali na više mjesta ćemo spomenuti i računsku i distribuiranu inteligenciju s obzirom na to da i nije baš jednostavno napraviti oštru podjelu između njih. Pregledno ih prikazuje slika 1-5.



Slika 1-5. Nebiološka ili apstraktna inteligencija (αI)

U čemu je razlika između osnovnih tipova nebiološke inteligencije?

Umjetna inteligencija usko je povezana s pojmom ***znanja***, njegovim prikupljanjem, pohranom u posebnim strukturama nazvanim baze znanja i primjenom tog znanja pri rješavanju kompleksnih problemskih zadataka. U sustavima umjetne inteligencije znanje je konstantno prisutno tijekom rada inteligentnog sustava i izdvojeno je od mehanizma pomoću kojega se ono primjenjuje pri rješavanju zadataka u strukture koju obično nazivamo ***baza znanja***. Ovakva se nebiološka inteligencija često naziva i ***klasična ili uobičajena umjetna inteligencija*** (engl. *Conventional Artificial Intelligence*) ili ***simbolička, čista ili logička umjetna inteligencija*** (engl. *Symbolic, Neat, Logical Artificial Intelligence*). Također je poznata je i pod već spomenutom kraticom ***GOFAI – Good Old Fashioned Artificial Intelligence*** što bismo slobodno mogli prevesti kao ***dobra staromodna umjetna inteligencija***.

Naziv GOFAI uveo je ***John Haugeland*** 1985. godine u knjizi „*Artificial Intelligence: The Very Idea”* u kojoj je razmatrao prvenstveno filozofske aspekte istraživanja u području umjetne inteligencije. ***Haugeland*** također uvodi i pojam ***sintetička inteligencija*** (engl. *Synthetic Intelligence*) kao alternativni naziv umjetne inteligencije kojim pokušava naglasiti da ovaj tip inteligencije nije imitacija biološke inteligencije, a nije ni umjetna. Ona je stvarna forma inteligencije dobivena sintezom od sastavnih dijelova.

Kako je ***Haugeland*** i kolokvijalno kazao, ona je verzija inteligencije koju je stvorio čovjek za razliku od biološke inteligencije koja se razvijala prirodno. Postupci klasične umjetne inteligencije uglavnom se uspješno koriste kod:

- ***Stručnih (ekspertnih) sustava*** (engl. *Expert Systems*) kod kojih se na temelju upita, baze znanja i procesa zaključivanja donosi odgovor ili zaključak. Primjer su razni dijagnostički sustavi kod kojih na temelju simptoma, pohranjenog znanja i mehanizma zaključivanja zaključujemo na uzroke.
- ***Zaključivanja na temelju slučajeva*** (engl. *Case-base Reasoning*) kod kojega se proces rješavanja zadatka temelji na rješenjima sličnih zadataka u prošlosti. Što je veća baza znanja prethodnih slučajeva to je veća vjerojatnost rješavanja i novog zadatka. Ovo je na neki način umjetna realizacija inteligencije koju Steinberg naziva kreativna ili iskustvena inteligencija.

U klasičnu umjetnu inteligenciju najčešće se uključuju i:

- ***Bayesove mreže*** (engl. *Bayesian Networks*) – strukture zaključivanja temeljene na direktnim acikličkim grafovima s pridruženim vjerojatnostima na temelju kojih se u pojedinim situacijama donose zaključci.

U svim ovim slučajevima znanje je uvijek prisutno ili u obliku izdvojenih formalnih baza znanja, u obliku baza slučajeva ili u obliku unaprijed određenih vjerojatnosti u Bayesovim mrežama. Neki autori u klasičnu umjetnu inteligenciju uključuju i umjetnu inteligenciju temeljenu na ponašanju:

- ***Umjetna inteligencija temeljena na ponašanju*** (engl. *Behavior Based Artificial Intelligence*) kod koje se radi modularno raščlanjivanje inteligencije na osnovne tipove ponašanja. Popularna je posebno u robotici, a primjer ove inteligencije je ***Brooksova supsumcijska***[4] (engl. *Subsumption*) arhitektura kod koje se složeno ponašanje razbija na više slojeva, od jednostavnijih ponašanja na dnu do složenijih ponašanja na vrhu, koji se paralelno izvode. U Brooksovoj supsumcijskoj arhitekturi znanje može i formalno biti prisutno u svakom sloju, ali je u biti stvarna primjena znanja u raščlanjivanju tipova ponašanja i formiranju pojedinih slojeva.

Računska inteligencija dolazi od riječi računati (engl. *Compute*), a ne računalo (engl. *Computer*). Zato se i zove računska, a ne računalna inteligencija kako je ponekad pojedini autori pogrešno nazivaju. Osnovna joj je značajka u tome da je znanje prisutno na početku kada se formira struktura pomoću koje ćemo rješavati zadatke, ili eventualno tijekom iterativnog razvoja strukture, ali kada je struktura jednom definirana, postupak rješavanja zadataka svodi se na čisto računanje. Poznata je i kao ***nesimbolička umjetna inteligencija*** (engl. *Non-symbolic Artificial Intelligence*), a za postupke koji se

[4] Značenje riječi *„supsumcijsko”* je *„podređivanje posebnog pod opće”*.

u okviru računske inteligencije upotrebljavaju koristi se i pojam ***meko računanje*** (engl. *Soft Computing*). U računsku inteligenciju uglavnom spadaju četiri vrste sustava:

- ***Umjetne neuronske (živčeve) mreže***[5] (engl. *ANN - Artificial Neural Networks*) pomoću kojih se nastoji kopirati sklopovski ustroj ljudskog mozga velikim brojem međusobno povezanih računskih jedinki koje nazivamo umjetni neuroni. Na temelju ulaznog znanja i postupka učenja formira se mreža, neke od veza između umjetnih neurona se prekidaju, a neke uspostavljaju. Na kraju procesa učenja mreža je sposobna (sa više ili manje uspjeha) dati odgovore i na upite koje nije imala tijekom procesa učenja. Danas se umjetne neuronske mreže često koriste u različitim područjima.
- ***Neizraziti sustavi*** (engl. *FS - Fuzzy Systems*) su razvijeni prije svega u svrhu zaključivanja uz prisustvo nesigurnosti i nepreciznosti, na način da nastoje oponašati logički sustav ljudskog mozga. Klasična umjetna inteligencija uglavnom se naslanja na klasičnu logiku i njena proširenja, a to sigurno nije logika ljudskog mozga koja u najvećem broju slučajeva donosi zaključak na temelju nesigurnih i nepreciznih ulaznih podataka. Tipičan primjer je vožnja automobila. Čovjek sigurno ne donosi zaključak tijekom vožnje tipa:

  „Ako je brzina automobila 78,6 km/h i udaljenost od automobila ispred 3,.4 m, a automobil vozi brzinom od 63,2 km/s treba stisnuti kočnicu silom od 56,8 N."

  Puno bliži opis našeg ponašanja pri vožnji automobila je iskaz:

  „Ako je brzina automobila prevelika u odnosu na brzinu automobila ispred, a razmak je mali, jako stisni kočnicu."

  Neizraziti sustavi bave se upravo ovakvim izjavama. Kako napraviti sustav čije se zaključivanje temelji na nepreciznim iskazima izraženim riječima i rečenicama prirodnog jezika. Kako realizirati računanje s riječima (engl. *CWW – Computing With Words*)? Ovim dijelom računske inteligencije, a posebno zaključivanjem u duhu neizrazite logike, bavimo se jednim dijelom i u ovom udžbeniku.



Slika 1-6. Jedan od prvih primjera primjene neizrazitih sustava u automatskom vođenju je Sendai Subway 1000[6] *željeznica iz Japana kod koje je neizraziti regulator korišten za regulacije brzine vlaka. Rezultat njegove primjene je udobnija vožnja i 10% ušteda energije.*

[5] Umjetne neuronske mreže ponekad se u hrvatskoj varijanti pogrešno nazivaju umjetne neuralne mreže na temelju pogrešnog prijevoda engleskog izraza „*Artificial Neural Networks*". Engleski naziv stanice po kojoj su i dobile naziv je „*Neural Cell*" koji je u hrvatskom prijevodu „*živac*" ili „*neuron*", pa je ispravan hrvatski naziv „*umjetne neuronske mreže*" ili eventualno manje korišteni naziv „*umjetne živčeve mreže*".

[6] Slika je s Web stranice https://en.wikipedia.org/wiki/Sendai_Subway_1000_series uz CC BY 3.0 licencu.

- ***Umjetni imunološki sustavi*** (engl. *AIS - Artificial Immune Systems*) temelje se na ***strojnom učenju potpomognutom pravilima*** (engl. *Rule-Based Machine Learning*) inspiriranim procesima imunološkog sustava kralježnjaka. Pojavili su se sredinom 1980-ih godina u istraživanjima imunoloških mreža, ali tek polovinom 1990-ih godina postaje samostalno istraživačko područje koje je predložilo niz algoritama inspiriranih imunološkim sustavima kralježnjaka. Primjeri su ***algoritam klonske selekcije*** (engl. *Clonal Selection Algorithm*), ***algoritam negativne selekcije*** (engl. *Negative Selection Algorithm*) i ***algoritmi imunoloških mreža*** (engl. *Immune Network Algorithms*).

- ***Evolucijsko računanje*** (engl. *Evolutionary Compution*) kod kojeg znanje na neki način i nije direktno prisutno, osim kroz mjere procjene kvalitete rješenja. Inspiraciju je pronašlo u evoluciji, a najpoznatiji postupak su ***genetski algoritmi*** (engl. *GA – Genetic Algorithms*) kod kojih se koriste pojmovi kao što su umjetni genom, populacija, mutacija, kros-korelacija, adaptacija, preživljavanje najboljih, itd. Često se koriste u svrhu optimizacije, na primjer optimizacija parametara regulatora, rješavanje zadataka traženja minimuma ili maksimuma u kompleksnim situacijama i slično.

Računska inteligencija danas je područje koje ima najširu komercijalnu primjenu. Brojni su uređaji široke potrošnje u kojima se nalaze neizraziti regulatori, a i primjena umjetnih neuronskih mreža iz dana u dan sve je veća, posebno pojavom dubokih neuronskih mreža i ***dubokog učenja*** (engl. *Deep Learning*). Uz duboko učenje, u posljednje se vrijeme pojavljuje i pojam ***utjelovljena inteligencija*** (engl. *Embodied Intelligence*) nastala naglim razvojem računske inteligencije i robotike s ciljem ugradnje inteligentnog ponašanja utjelovljenim agentima (različitim robotskim sustavima) kako bi mogli samostalno djelovati u određenom okružju svjesni svoga okružja i svojih ograničenja. Ovo je sigurno područje na kojem se u bliskoj budućnosti očekuju značajni doprinosi. Naglasimo još jednom da je znanje kod sustava temeljenih na računskoj inteligenciji obično prisutno samo na početku dok se struktura ili sustav formira, a nakon toga tijekom rada radi se o relativno jednostavnim računskim algoritmima.

Distribuirana umjetna inteligencija koju ponekad nazivaju i ***decentralizirana umjetna inteligencija*** svoju inspiraciju nalazi u biološkim sustavima ***kolektivne inteligencije*** (engl. *Collective Intelligence*) koju još 1983. godine spominje ***Peter Russel*** u svojoj knjizi „*Globalni mozak”* (engl. *„The Global Brain”*). Russel polazi od tzv. ***Gaia hipoteze***, ekološke teorije postavljene 1960. godine od strane ***Jamesa Lovelocka*** (1919. –) po kojem sva živa bića na zemlji čine jedinstveni kolektivni organizam, a završava s idejom o socijalnom inteligentnom superorganizmu. Kasnije su se na ovu teoriju nadovezali brojni autori, a osnovna je ideja da se inteligentno ponašanje može pojaviti kao rezultat interakcije velikog broja jedinki po principima ***suradnje*** (kooperacije) ili ***suparništva*** (kompeticije). Pri tome pojedine jedinke nemaju nikakva posebna svojstva inteligencije. Školski primjeri ovakvih sustava su kolonije mrava ili rojevi pčela prikazani na slici 1-7.

Poznato je da je primjerice u termitnjacima sistem prirodnog provjetravanja toliko savršeno izveden da se ima dojam da ga je „projektiralo” inteligentno biće, a u biti je nastao kao rezultat kolektivnog rada vrlo primitivnih jedinki. To je i temeljno polazište distribuirane umjetne inteligencije kod kojeg se jedinke realiziraju kao programski moduli ili kao robotski sustavi. Ponekad se ovaj tip inteligencije naziva i ***inteligencija gomile (mnoštva)*** (engl. *Swarm Intelligence*), a neki je autori smatraju dijelom računske inteligencije. U posljednje se vrijeme distribuirana umjetna inteligencija naslanja i na područje ***agentskih*** (engl. *Agent Systems*) i ***višeagentskih*** (engl. *Multi-Agent Systems*) sustava.

Tri su osnovna smjera istraživanja unutar distribuirane umjetne inteligencije:

- ***Paralelno rješavanje zadataka*** sa ciljem prilagodbe koncepata razvijenih unutar umjetne inteligencije pomoću sustava s više jezgri i pomoću klastera računala.

- ***Distribuirano rješavanje zadataka*** u kojemu se pojedini dijelovi zadatka daju nezavisnim entitetima (agentima) koji ih samostalno rješavaju, a sva pojedinačna rješenja doprinose globalnom rješenju.

- ***Višeagentske simulacije*** kod kojih se u simulacijama ne modeliraju skupine već pojedine jedinke, na primjer ekološko modeliranje kod kojega se ribe ne modeliraju kao generalni

kolektivitet ribe koji u model ulazi kao jedna varijabla, već se svaka pojedina riba predstavlja jednim entitetom (agentom) a cijeli se model temelji na interakciji milijuna takvih entiteta. Najveća primjena ovakvih sustava je u računalnim igrama.



Slika 1-7. Ponašanja kolonije mrava bila su poticaj za istraživanja distribuirane inteligencije. Na slici je primjer kooperacije – mravi tijelima izgrade most kako bi došli do područja koja im nisu direktno dohvatljiva[7]

Na kraju naglasimo da distribuirana umjetna inteligencija ima poseban odnos u odnosu na znanje. Znanje u sustavima distribuirane inteligencije nije nigdje posebno izdvojeno i pohranjeno kao kod sustava umjetne inteligencije, niti je na temelju njega formirana nekakva struktura kao kod računske inteligencije, već je znanje skriveno u zakonima interakcije između pojedinih jedinki.

Posebno područje distribuirane umjetne inteligencije su ***kolektiviteti inteligentnih jedin*ki** (inteligentni višeagentski sustavi) koji kao svoju prirodnu inspiraciju imaju ***civilizaciju***. Civilizacija nastaje kao rezultat kolektivnog rada inteligentnih jedinki (ljudi). U ovom slučaju znanje je pohranjeno i u svakoj pojedinoj jedinki, ali i skriveno u interakcijama između tih inteligentnih jedinki

Spomenimo još i pojam ***umjetnog života*** (engl. *ALife ili A-Life*) koji je usko vezan uz agentske i višeagentske sustave. Naziv uvodi 1986.g. teorijski biolog ***Christopher Langton*** (1948./49.-), iako su se i prije njega brojni istraživači bavili ovom tematikom. Primjer je kibernetičar ***Valentino Breitenberg*** (1926. – 2011.) koji 1984.g. u svojoj knjizi o vozilima (engl. *Breitenberg Vehicles*) (Breitenberg, 1984.) modelira životinjski svijet na jednostavan, minimalistički način koristeći samo reaktivno ponašanje. Od tada do danas, brojni istraživači su se bavili ovom problematikom[8]. Osnovno ideja ovih istraživanja jest da se prije inteligencije pojavio život, pa bi možda prvi korak u istraživanju umjetne inteligencije trebalo biti proučavanje i realizacija virtualnih ili fizičkih (tehničkih) naprava sa značajkama života. Proučavaju se značajke života vezane sa živim bićima na različitim stupnjevima razvoja, od virusa na granici živog i neživog, pa do inteligentnih bića. Nastoje se replicirati osnovne značajke života, koje prije svega uključuju sposobnost ***nutrifikacije*** (hranjenja), ***reprodukcije*** (razmnožavanja) i ***adaptacije*** (evolucijske prilagodbe) koja je primjerice kod životinjama, za razliku od biljaka, razvila i mogućnost ***lokomocije*** (kretanja) koja olakšava pronalaženje hrane i partnera za reprodukciju. Rezultati istraživanja umjetnog života danas su posebno važni u filmskoj industriji i industriji računalnih igara.

[7] Slika je s Web stranice https://www.jenal.org/2012/01/ uz CC BY-SA 3.0 licencu.

[8] Za više detalja vidi stranice udruge ***International Society for Artificial Life*** - https://alife.org

U nastavku ovog udžbenika uglavnom se bavimo klasičnom (simboličkom) umjetnom inteligencijom koja je povijesno najstarija i temelj je za razumijevanje ostalih postupaka, ali ćemo na mnogim mjestima spominjati i računsku i distribuiranu umjetnu inteligenciju.

1.3 Formalne definicije umjetne inteligencije

Još uvijek ne postoji jedinstvena definicija umjetne inteligencije. Svaki je autor nastoji definirati na svoj način, a neke od postavljenih definicija su:

- Umjetna inteligencija je znanost čiji je cilj napraviti umjetnu tvorevinu - stroj sposoban rješavati zadatke za čije rješenje je potrebna inteligencija ako ih rješava čovjek. (***Marvin Minsky***, MIT)
- Umjetna inteligencija nastoji oponašati ljudski način mišljenja i ljudske kognitivne procese, kako bi riješila složene probleme. (***Richard Stottler***, Sttotler Henke Ass.)
- Umjetna inteligencija proučava postupke računanja koji bi omogućili percepciju, zaključivanje i djelovanje. (***Patrick Winston***, MIT)
- Umjetna inteligencija je dio računalnih znanosti čiji je cilj istraživanje simboličkog, nealgoritamskog procesa rezoniranja i prikazivanja simboličkog znanja te njihova primjena inteligentnim strojevima. (***Edward Feigenbaum***, Stanford University)
- Umjetna inteligencija je područje računarstva povezano s razumijevanjem prirode inteligentnih postupaka i konstruiranjem umjetnog sustava sposobnog provoditi takve postupke. (***McGraw Hill Encyclopedia of Science & Technology***)
- Inteligencija (misli se na umjetnu inteligenciju) je sposobnost sustava da djeluje u nepoznatom okružju u kojem je primjereno djelovanje ono koje povećava vjerojatnost uspjeha, a uspjeh je postignuće pod-ciljeva koji potpomažu konačan cilj. (***Alexander Meystel i James Albus***, *Intelligent Systems – Architecture, Design and Control,* 2002.)
- Umjetna inteligencija je znanost kojoj je cilj napraviti stroj - računalo sposobno obavljati postupke koje u ovom trenutku čovjek obavlja bolje. (***Elain Rich***, University of Texas, Austin)

Ova posljednja definicija možda je najjednostavnija, ali i najprimjerenija. Sagledamo li područja kojima se umjetna inteligencija bavi, zaključuje se da je osnovni cilj umjetne inteligencije napraviti stroj koji će se ponašati kao čovjek. Pri tome nije bitno da se u potpunosti kopiraju načini na koje čovjek pohranjuje i primjenjuje znanja, pristupa rješavanju zadataka i slično, već je bitno da konačan rezultat, rješenje nekog zadatka, bude takvo kao da ga je riješio čovjek ili još bolje od čovjekovog rješenja.

Postoje dva različita motiva zbog kojih se umjetna inteligencija uopće razvija kao znanost. Kod prvoga se kaže da se umjetna inteligencija treba razvijati kako bi se bolje razumjela ljudska inteligencija. Kod drugoga umjetnu inteligenciju treba razvijati kako bi se jednostavnije rješavali određeni kompleksni zadaci koji su sada teško rješivi i za čije je rješenje nužan čovjek. Ovaj zadnji motiv zbog kojeg se istražuje umjetne inteligencije posebno ističemo i dodatno dorađujemo, na možda malo provokativan način kako bismo istaknuli i vrijeme u kojem živimo u kojemu je često najvažnija stvar profit. Jedan od motiva za istraživanja umjetne inteligencije, ali i svih ostalih inteligentnih tehnologija je svakako:

- zamijeniti ljudsku inteligenciju na određenim zadacima, zato što je čovjek zahtijevan, skup i nepouzdan.

Ima mnogo primjera koji su potpora ovoj tvrdnji. Tipičan primjer je klasični primjer uvođenja inteligentnog neizrazitog vođenja kod vođenja rotacijske peći za proizvodnju cementa. Proizvodnja cementa je kontinuiran proces koji traje 24 sata i toliko kompleksan da su svi pokušaji automatizacije procesa uobičajenim postupcima automatskog vođenja ostali bez uspjeha. Rotacijsku peć vode tri operatera u tri smjene, a svaki od njih nije jednako sposoban niti uvijek maksimalno koncentriran. Na primjer kada u Dalmaciji zapuše jugo, koncentracija operatera automatski pada. Rezultat je taj da proces proizvodnje cementa nije u svakom trenutku optimalan. Kvaliteta je obično unutar normi, ali često varira potrošnja goriva. Čovjek operater je poslodavcu skup, troši više goriva negoli je to potrebno. Inteligentno

vođenje rotacijske peći neizrazitim regulatorom temeljilo se na izvlačenju znanja o vođenju peći najboljeg operatera u trenucima maksimalne koncentracije. Efikasan postupak kodiranja jezičnih pravila vođenja u matematički oblik koji je osigurala neizrazita logika rezultirao je inteligentnim regulatorom kojem jugo nije smetalo. Rezultati primjene ovakvog regulatora bili su i do 10% manja prosječna potrošnja goriva što se u cementnoj industriji mjeri velikim svotama novca. Autori su se hvalili da se investicija uložena u razvoj inteligentnog vođenja rotacijske peći za proizvodnju cementa temeljenog na neizrazitoj logici isplatila u par godina. Istina je da čovjek operater nije potpuno zamijenjen i uklonjen iz upravljačke sobe. On je i dalje ostao spreman da intervenira kada nešto krene pogrešno. Maksimalno je koncentriran na taj zadatak, a rutinsko vođenje prepušteno je inteligentnom računalnom sustavu. Zbog toga ćemo malo ublažili prethodnu izjavu u kojoj smo naveli zbog čega se umjetna inteligencija izučava te kazati da se umjetna inteligencija izučava kako bismo:

- zamijeniti ljudsku inteligenciju i čovjeka na određenim zadacima koji su za čovjeka zamorni, monotoni, teški ili opasni.

Kao i svaka druga suvremena znanost tako i umjetna inteligencija ima svoje protivnike i pristaše, no bez obzira je li se netko svrstava u pristaše ili protivnike, osnovni se principi trebaju poznavati kako bi se moglo argumentirano napadati ili braniti. Tipični komentar skeptika istraživanja inteligentnih tehnologija je:

„Pričali vi što pričali, ali računalo može napraviti samo ono što mu se kaže da napravi."

Ako je ova tvrdnja točna, kako se onda može očekivati inteligencija od stroja koji samo slijedi instrukcije? Odgovor leži u interpretaciji tvrdnje. Računalo izvodi program korak po korak, ali u programima umjetne inteligencije nije moguće predvidjeti kako će se program ponašati pod utjecajem nekih od velikog broja mogućih uvjeta i stimulansa za koje se niti ne zna hoće li se pojaviti, kada će se pojaviti i kakvi će biti. Situacija je slična odnosu učitelj – učenik. Učitelj nastoji pripremiti učenika za sve situacije u životu koje on može predvidjeti, ali kako će se učenik stvarno ponašati, a pogotovo u novoj situaciji, to on ne može garantirati. Nadalje, programi obično imaju i mogućnost učenja i samokorekcije, pa je ishod djelovanja inteligentnog sustava potpuno nepoznat u trenutku njegovog kreiranja.

Druga vrsta kritike koja se zna često čuti je:

„Dobro, računalo može komponirati glazbu, ali usporedimo li tu glazbu s glazbom Mozarta, onda ona nije dostojna ni slova M.", ili

„Računalo može dokazivati teoreme Euklidove geometrije, ali nikada ne može izmisliti Euklidovu geometriju."

„To je istina", kažu zagovornici umjetne inteligencije, *„ali koliko je u ljudskoj povijesti bilo Euklida i Mozarta? Koliko je godina trebalo proći da se od početka Homo sapiensa rodi čovjek sposoban izmisliti geometrijske teoreme? Usporedimo li to sa samo 60 godina prošlosti umjetne inteligencije, kako se onda može tvrditi da nešto nikada neće biti moguće?"*

Što će biti, bit će. Nas u ovom trenutku umjetna inteligencija prvenstveno zanima kao inženjerska disciplina, alat pomoću kojega ćemo biti u mogućnosti riješiti određene probleme u različitim područjima ljudske djelatnosti koji su do sada jedino uspješno rješavali ljudi. Spomenimo još na kraju ovog uvodnog poglavlja da postoje dva temeljna pristupa istraživanjima umjetne inteligencije. Prvi pristup je tzv. ***jaka umjetna inteligencija*** (engl. *Strong AI*) ili kako se ponekad naziva ***generalna umjetna inteligencija*** (engl. AGI – *Artificial General Intelligence*) kojoj je cilj razviti stroj (umjetnu tvorevinu) koji će imati iste intelektualne sposobnosti kao i čovjek. Drugi pristup je ***slaba umjetna inteligencija*** (engl. *Weak AI*) ili ***uska umjetna inteligencija*** (engl. ANI – *Artificial Narrow Intelligence*) koja se fokusira na jedan uski zadatak koji nastoji riješiti na način kako ga rješavaju ljudi. U biti *svi do sada realizirani sustavi umjetne inteligencije* spadaju u slabu umjetnu inteligenciju. MYCIN je bio ekspertni sustav usmjeren na dijagnosticiranje bakterijskih infekcija i bolesti zgrušavanja krvi, a nije bio univerzalni dijagnostičar koji detektira sve bolesti. Teoretičari se ne slažu je li jaku umjetnu inteligenciju uopće moguće realizirati, a povezano s tim možemo spomenuti i tzv. ***argument znanja*** (engl. *Knowledge Argument*) poznat i pod nazivom ***Marijina soba*** (engl. *Mary's Room*) (slika 1-8).

Radi se o filozofskom (misaonom) eksperimentu koji je 1982. godine uveo australski analitički filozof ***Frank Cameron Jackson*** (1943. –). Ideja eksperimenta bila je pokušati pokazati da mentalna

stanja nemaju fizička svojstva. Priča ide ovako: „*Marija je neuroznanstvenica koja cijeli svoj život živi u prostoriji bez boje – nikada nije izravno iskusila boju u svom cijelom životu, iako je za to sposobna. Kroz crno-bijele knjige i druge medije, ona se obrazuje o neuroznanosti do te mjere da postaje stručnjak za temu boja. Marija zna sve što se može znati o percepciji boje u mozgu, kao i o fizikalnim modelima kaje povezuju valne duljine i boju. Marija je svjesna svih fizikalnih činjenica o boji i percepciji boje. Jednog dana Marija izlazi iz sobe i doživljava, po prvi put, izravnu percepciju boja. Prvi put vidi crvenu boju i sazna nešto novo o tome kako crvena izgleda. Jackson je zaključio da pristup opisu svijeta nazvan **fizikalizam**, koji smatra da nema ničega izvan i iznad fizičkoga, nije istinit jer bi Marija u tom slučaju trebala steći ukupno znanje o percepciji boje samim skupljanjem znanja o fizikalnim događanjima vezanim s bojom i percepcijom boje. Međutim, budući da je Marija nešto naučila o boji tek kada je napustila sobu i vidjela boju, onda fizikalizam mora biti lažan.*"

Jacksonov eksperiment možemo poopćiti i na područja umjetne inteligencije. Sustav umjetne inteligencije može prikupljati podatke o nečemu iz fizičkog svijeta na isti način kao i Marija, ali bez stvarnog doživljaja fizičkog svijete, za što on nije sposoban, nema te dodatne dimenzije koju je Marija spoznala kada je vidjela crvenu boju.



Slika 1-8. Ilustracija misaonog eksperimenta Marijina soba[9]

Neki od znanstvenika ovo smatraju filozofskim argumentom protiv jake umjetne inteligencije, a u korist slabe. Smatraju da se treba koncentrirati na pojedini manji zadatak i u okviru njega napraviti nešto korisno, kao u primjeru Sendai željeznice kod koje pametni sustav vođenja regulira brzinu na isti način kako bi to radio dobro obučen ljudski strojovođa. Pri tome nije nužno točno oponašati ono što strojovođa radi, bitno je dobiti isti rezultat.

[9] Slika je s https://commons.wikimedia.org/wiki/File:Mary_colour_scientist.png uz CC BY-SA 3.0 licencu.

Spomenimo na kraju još pojam ***umjetne superinteligencije*** *(*engl. *ASI – Artificial Super Intelligence)* koji je, na našu sreću, sada još uvijek samo tema knjiga i filmova znanstvene fantastike. Podrazumijeva umjetnu inteligenciju koja bi bila superiornija u odnosu na ljudsku inteligenciju, nešto kao *„Terminator"* kojeg je utjelovio ***Arnold Schwarzeneeger*** . Umjetna superinteligencija u knjigama i filmovima na ovu temu uglavnom vodi prema izumiranju ljudskog roda. U ovom smo trenutku na našu sreću, još daleko i od generalne umjetne inteligencije, pa će nas umjetna superinteligencija i dalje samo zabavljati s filmskog platna.

1.4 Područja istraživanja umjetne inteligencije

Čime se sve bavi umjetna inteligencija? Vratimo se opet malo u prošlost i početke istraživanja umjetne inteligencije. Prvi problemi koje bismo mogli svrstati u probleme umjetne inteligencije su problemi igranja igre i dokazivanja teorema. Godine 1963. napisana su dva programa. Jedan od njih ne samo da je igrao igru, već je i poboljšavao svoje kvalitete tijekom igranja, a drugi je mogao dokazivati osnovne teoreme iz knjige filozofa i matematičara ***Bertranda Russella*** *„Principia Matematica"*. Negdje u isto vrijeme javljaju se i prvi pokušaji gradnje programa koji bi mogli na univerzalni način rješavati zadatke, nešto kao ***univerzalni rješavač problema*** (engl. *General Problem Solver*). Početni zadaci bili su dosta jednostavni i usmjereni prema problemima koje čovjek rješava svakog dana pri odlučivanju kako ići na posao.

Kako su istraživanja napredovala tako su se razvijali i postupci za manipuliranje velikim količinama znanja, jer, kako ćemo kasnije naglasiti, sakupljanje, prikazivanje, pohrana i manipuliranje znanjem jedna je od osnovnih karakteristika sustava umjetne inteligencije. To se područje danas razvija kao nezavisno područje poznato pod pojmom ***inženjerstvo znanja*** (engl. *Knowledge Engineering*). Posebna područja unutar inženjerstva znanja odnose se na istraživanja (slika 1-9):



Slika 1-9. Osnovna područja inženjerstva znanja

- ***percepcije*** koja uključuje prepoznavanje slike i objekata na njoj, govora, mirisa, okusa, itd.
- ***razumijevanja*** i to posebno razumijevanje govora, slike, scene itd.
- ***učenja*** koje može biti samostalno ili potpomognuto i
- ***rješavanje zadataka*** za čije rješenje je potrebno znanje pa ga obično nazivamo ***ekspertno rješavanje zadataka***.

U okviru ovog udžbenika najprije ćemo se baviti upravo tom problematikom rješavanja zadataka i to prije svega rješavanjem problemskih zadataka za čije je rješavanje potrebno znanje. U prvom dijelu to će biti općenite, univerzalne metode rješavanja zadataka i prikazivanja znanja. U drugom dijelu posvećujemo se temeljnim postupcima strojnog učenja koje je danas vrlo značajno prije svega zbog toga što je pronašlo veliku komercijalnu primjenu. Percepcijom i razumijevanjem se u ovom udžbeniku ne bavimo.

Unutar istraživanja povezanih s inteligentnim tehnologijama možemo definirati i nekoliko kategorija u odnosu na moguće primjene. Uveli su ih 1983. godine ***Frederick Hayes-Roth, Donald Waterman*** *i* ***Douglas Lenat*** (Hayes-Roth, Waterman i Lenat, 1983.):

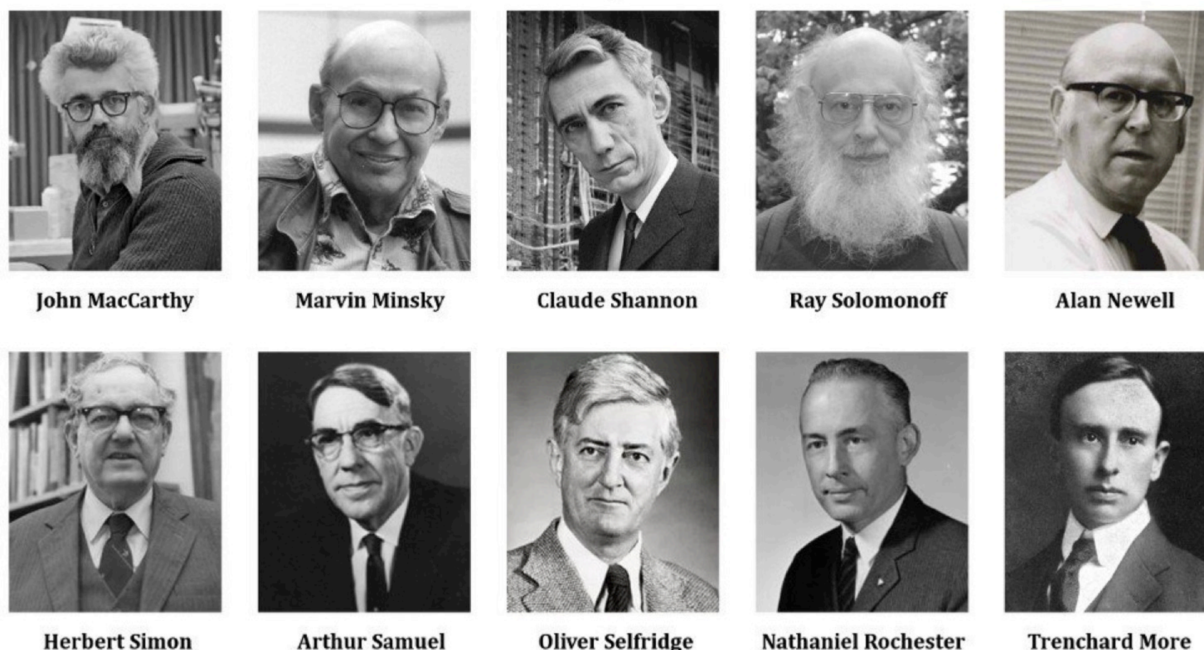
- ***Interpretacija*** (engl. *Interpretation*) – zaključivanje o situaciji na osnovi informacija sa osjetila (senzora), tj. kombinacija percepcije i razumijevanja. Na primjer, razumijevanje govora, analiza slike, interpretacija signala, rekonstruiranje kemijske strukture.
- ***Predviđanje*** (engl. *Prediction*) – predviđanje mogućih posljedica na temelju dane situacije, na primjer predviđanje vremena, demografska predviđanja, procjena uroda, vojna predviđanja ishoda sukoba.
- ***Dijagnostika*** (engl. *Diagnosis*) – zaključivanje o mogućim greškama na temelju opažanja (opservacija), na primjer medicinska dijagnostika, dijagnostika digitalnih sklopova, dijagnostika mehaničkih sustava.
- ***Projektiranje*** (engl. *Design*) – konfiguriranje objekata uz prisustvo ograničenja, na primjer projektiranje elektroničkih sklopova, projektiranje štampanih veza, urbanističko projektiranje, investicijsko projektiranje.
- ***Planiranje*** (engl. *Planning*) – projektiranje aktivnosti i slijeda aktivnosti, na primjer automatsko programiranje, planiranje vojne akcije, planiranje građevinskih radova.
- ***Nadgledanje*** (engl. *Monitoring*) – uspoređivanje opažanja (opservacija) s osjetljivošću pogona, na primjer nadgledanje pogona, nadgledanje rada atomskih centrala, nadgledanje zračnog prometa, nadgledanje širenja zarazne bolesti.
- ***Otkrivanje grešaka*** (engl. *Debugging*) – postupci pronalaženja grešaka, na primjer u pisanom tekstu ili računalnom programu.
- ***Održavanje*** *(engl. Maintenence)* – izvršavanje plana provjere i povremene i preventivne zamjene, na primjer održavanje računalnih sustava, održavanje plovila vozila i letjelica, održavanje strojeva, održavanje zdravlja.
- ***Podučavanje*** (engl. *Instruction*) – prenošenje znanja ili vještina, dijagnosticiranje primljenog i prihvaćenog znanja ili vještina, otkrivanje nedostataka *(„rupa”)* i ispravljanje tih nedostataka, na primjer u sustavu obrazovanja i obuke (inteligentni tutorski sustavi).
- ***Vođenje*** (engl. *Control*) – interpretiranje, predviđanje, nadgledanje i održavanje osmišljenog suvislog ponašanja sustava, na primjer procesno vođenje, vođenje zadatka ili misije, vođenje prometa, vođenje bitke.

Ovo su neki od osnovnih područja kod kojih se dosta primjenjuju inteligentni postupci i metode. Sigurno ih ima još, a u budućnosti će ih sigurno biti još i više. Prije negoli krenemo na detaljnu razradu postupaka i metoda koje koristi umjetna inteligencija i njoj srodne tehnologije, okrenut ćemo se malo prošlosti i dati kratku povijest umjetne inteligencije i inteligentnih tehnologija. Nakon toga ćemo se osvrnuti na problematiku kako utvrditi jesmo li postigli cilj i stvarno sagradili program ili uređaj koji ima neko od svojstava inteligencije.

1.5 Kratka povijest umjetne inteligencije

Umjetna je inteligencija relativno mlada znanstvena disciplina koja se naslanja na znanja do kojih se došlo stoljetnim istraživanjima u područjima filozofije, matematike, psihologije, neurologije, kibernetike i suvremenog računarstva. Složenost područja umjetne inteligencije je upravo u tome što je multidisciplinarna i ujedinjuje znanja iz područja tehničkih, društvenih i bioloških znanosti.

Formalno rađanje umjetne inteligencije kao posebne znanstvene discipline i formiranje naziva ***AI – Artificial Intelligence*** koji prevodimo ***umjetna inteligencija*** dogodilo se 1956. godine na Dartmouth Collegeu. ***John McCarthy*** (1927. – 2011.) (koji je na tom koledžu i radio), ***Marvin Minsky*** (1927. – 2016.), ***Claude Shannon*** (1916. – 2001.) i ***Nathaniel Rochester*** (1919. – 2001.) organizirali su dvomjesečnu radionicu na kojoj se skupilo 10 ljudi koji su se bavili teorijom automata, neuronskim mrežama i istraživanjem inteligencije (slika 1-10).



Slika 1-10. Deset sudionika radionice na Dartmouth Collegeu 1956. godine kada je, za tada novu znanstvenu disciplinu, na prijedlog John McCarthya usvojeno ime AI - Artificial Intelligence – umjetna inteligencija[10]

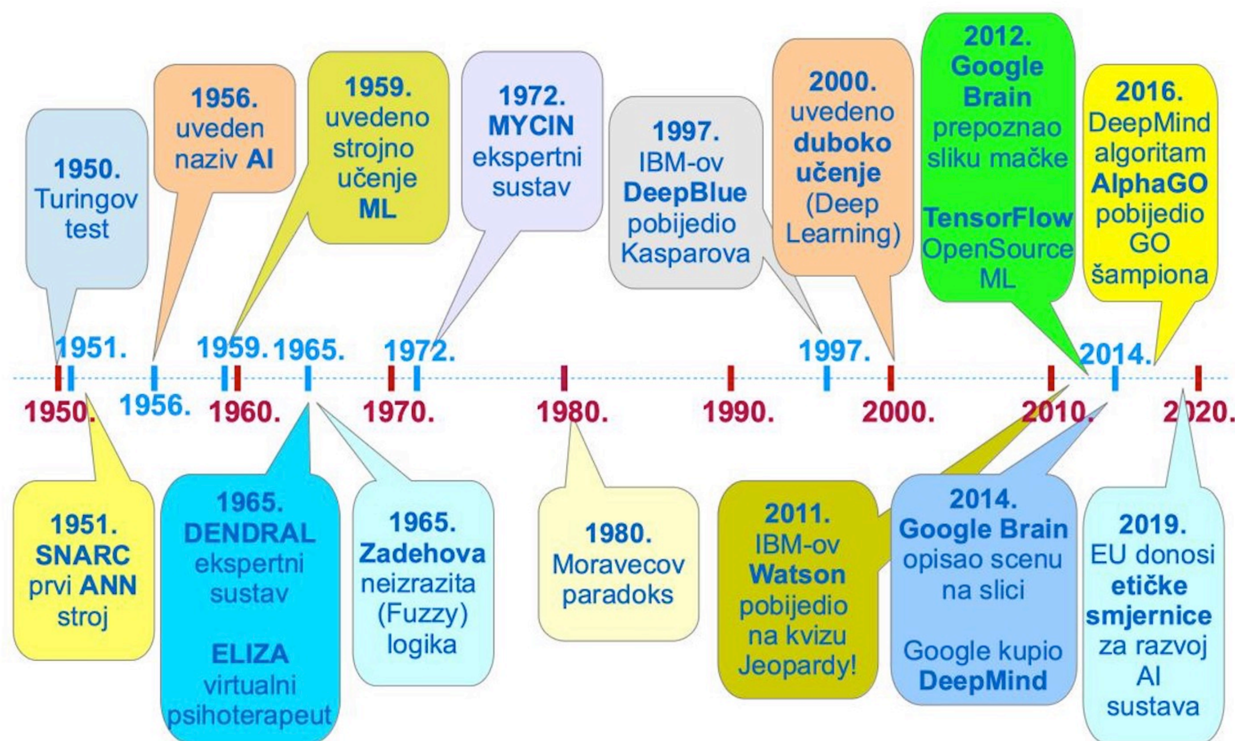
Na radionici su sudjelovali i ***Trenchard More*** sa Sveučilišta Princeton, ***Arthur Samuel*** (1901. – 1990.) iz IBM-a, ***Ray Solomonoff*** (1926. – 2009.) i ***Oliver Selfridge*** (1926. – 2008.) s MIT-a te ***Allen Newell*** (1927. – 1992.) i ***Herbert Simon*** (1916. – 2001.) s Carnegie Techa (danas Carnegie Mellon). Sudionici radionice prihvatili su prijedlog Johna McCarthyja da se novo područje istraživanja nazove ***umjetna inteligencija*** *(engl.* ***Artificial Intelligence***) iako su smatrali da bi naziv ***računska racionalnost*** (engl. *Computational Rationality*) možda bolje odgovarao.

Nekoliko važnijih događanja vezanih uz povijest umjetne inteligencije prikazano je na slici 1-11.

Da se ne vraćamo puno u povijest i mehaničke automate kojima su vrsni majstori pokušati oponašati živa bića, suvremena umjetna inteligencija počinje 1950. godine s ***Alanom Turingom*** (1912. – 1954.), pionirom računarstva koji te godine definira poznati Turingov test pomoću kojega se testira koji je stupanj inteligencije postigao stroj (više o njemu na kraju ovog 1. poglavlja). Godinu dana kasnije, 1951. godine, ***Marvin Minsky***, jedan od sudionika ljetne škole na Dartmouth Collegeu, sagradio je ***SNARC***

[10] Slika je sa stranice https://www.slideshare.net/hys_enterprise/artificial-intelligence-in-software-testing

(Stochastic Neural Analog Reinforcement Calculator), prvo računalo temeljeno na umjetnim neuronskim mrežama, prvi pionirski napor u području koje je dobilo ime umjetna inteligencija.



Slika 1-11. Važnija događanja u povijesti umjetne inteligencije

Stuard Russell i ***Peter Norvig*** u svojoj knjizi *„Artificial Intelligence – A Modern Approach"* (Russel, Norvig, 2009.), koja je sigurno jedan od najboljih udžbenika umjetne inteligencije, cijelu povijest umjetne inteligencije dijele u sedam razdoblja:

1. faza ranog entuzijazma i velikih očekivanja (1950. – 1969.)
2. faza realnosti (1966. – 1973.)
3. faza sustava temeljenih na znanju (1969. – 1979.)
4. faza kada umjetna inteligencija postaje industrija (1980. – do danas)
5. faza povratka umjetnih neuronskih mreža (1986. – do danas)
6. faza umjetne inteligencije kao znanosti (1987. – danas)
7. faza inteligentnih agenata (1995. – do danas).

*U **fazi ranog entuzijazma i velikih očekivanja*** postavljeni su brojni zadaci, do tada nerješivi, za koje su istraživači (i to uglavnom oni koji su se sastali na radionici 1956. godine u Darthmouth koledžu) smatrali da će ih umjetna inteligencija moći riješiti. Treba imati na umu da je to doba kada je i cijelo računarstvo u svojim začecima. Do pojave istraživanja u području umjetne inteligencije smatralo se da je računalo samo malo bolji stroj za zbrajanje. Spomenimo samo slavni sustav ***Newella i Simona*** za univerzalno rješavanje zadataka – ***General Problem Solver*** koji je od početka projektiran na način da je kopirao ljudski postupak rješavanja zadataka. U IBM-u je ***Herbert Gelenter*** 1959. godine napisao računalni program za dokazivanje geometrijskih teorema – ***Geometry Theorem Prover***. Možda najveći napredak tog doba bilo je definiranje programskog jezika ***Lisp*** koji je razvio ***McCarthy*** 1958. godine kada se prebacio s Dartmouth koledža na MIT. Lisp je postao dominantni jezik programiranja zadataka umjetne inteligencije. Počinje se raditi i na prvim stručnim ili ekspertnim sustavima, primjerice početkom 60-ih godina počinje se razvijati poznati ***DENDRAL*** čiji je zadatak bio odrediti molekularnu strukturu

na temelju informacija dobivenih od spektrometra mase i znanja iz kemije. Pri tome se intenzivno koristilo znanje pohranjeno u obliku velikog broja produkcijskih pravila oblika. Slijedio ga je 1972. godine isto tako slavni ***MYCIN*** ekspertni sustav za dijagnostiku bakterijskih infekcija i preporuku antibiotika i njegovog doziranja u ovisnosti o težini pacijenta. Temeljio se na 600 pravila, a rezultati su bili bolji od rezultata dijagnostike koju bi dao mlađi doktor. Tu je i ***ELIZA*** razvijena na MIT-u, prvi program za obradu prirodnog jezika koji je glumio psihoterapeuta. ***Lotfi Zadeh*** 1965. godine postavlja temelje neizrazite logike, ali se u ovoj prvoj fazi još nije vidjelo koji će ona imati utjecaj na primjenu inteligentnih tehnologija u svakodnevnom životu.

Tijekom početne entuzijastičke faze istraživanja istraživači su obećavali puno više negoli su uspjeli napraviti. Poznata je izjava ***Simona*** da će za 10 godina računalo postati šahovski velemajstor. To se međutim nije ostvarilo, ali se ostvarilo nakon 40 godina, tako da je nakon faze velikih očekivanja slijedila ***faza realnosti*** u kojoj se nastavilo istraživati, ali možda ne više uz toliko spektakularne najave.

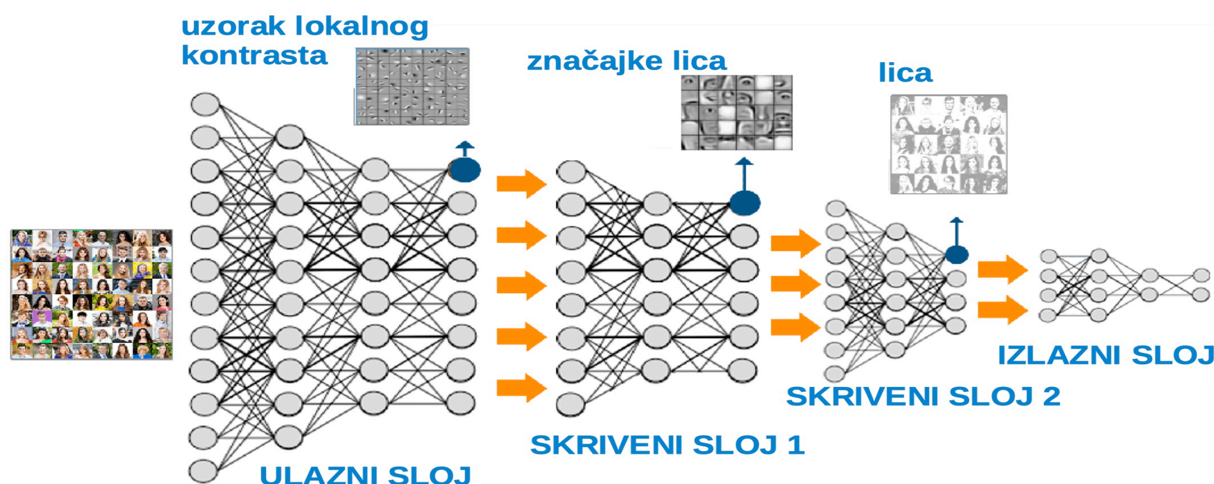
Formulirale su se metode rješavanja zadataka koje su nazvane ***slabe metode*** o kojima govorimo u sljedećem poglavlju, a istraživači su postali svjesni činjenice da je ključan element sustava umjetne inteligencije ***znanje*** (engl. *Knowledge*), ali isto tako da sustav koji ima pohranjeno znanje ne mora nužno pokazivati sva svojstva inteligencije o kojem se govorilo u prvoj fazi razvoja. Uvodi se pojam ***sustava temeljnih na znanju*** (engl. *KBS – Knowledge Based Systems*) kao šire i općenitije kategorije, pa se istraživanja šire na problematiku prikupljanja, formalne pohrane i pretraživanja znanja s ciljem rješavanja određenog kompleksnog problema. Zbog toga se ova faza i zove ***faza sustava temeljenih na znanju***. Ovo je doba kada se intenzivno razvija i računalna lingvistika, a važna je i po tome što je 1975. godine ***Minsky*** postavio ideju pohrane znanja u ***okvirima*** (engl. *Frames*), a 1970. godine ***Colmerauer*** i njegovi suradnici sa Sveučilišta u Marseille-Aix uvode novi programski jezik nazvan ***PROLOG*** (*PROgraming in LOGic*) koji vrlo brzo uz Lisp postaje temeljni jezik umjetne inteligencije. Na Queen Mary College u Londonu 1975. godine ***Mamdani i Assilian*** realizirali su prvu praktičnu primjenu neizrazite logike u automatskom vođenju (engl. *Fuzzy Control*) nakon čega slijedi intenzivni razvoj ovog područja i masovna primjena.

Osamdesete su doba kada se umjetna inteligencija komercijalizira pa počinje ***faza kada umjetna inteligencija postaje industrija*** koja traje do danas. Na tržištu je sve više komercijalnih proizvoda – često se ističe primjer tvrtke *Digital Equipment Corporation* koja je 1986. godine primijenila stručni sustav na konfiguriranju novih računala i uštedjela (prema procjeni kompanije) više od 40 milijuna američkih dolara. Skoro svaka veća kompanija u Sjedinjenim državama osniva svoju grupu za umjetnu inteligenciju, pa je primjerice *Du Pont* koristio preko 100 stručnih sustava (a razvijao 500 novih) koji su mu godišnje donosili 10 milijuna američkih dolara uštede. Japanci najavljuju razvoj tzv. Pete generacije računala koji bi trebali uključivati jezike umjetne inteligencije, a ekonomski stručnjaci procjenjuju da je umjetna inteligencija kao industrija 1980. godine počela s godišnjim obrtom sredstava u vrijednosti od nekoliko milijuna dolara, da bi već 1988. godine dosegla preko milijardu.

S jedne strane umjetna inteligencija postaje industrija, ali se otvaraju i brojna nova pitanja. ***Hans Moravec*** (1948. –) s *Instituta robotike Carnegie Mellon University* 1980. godine otkrio je paradoks koji se po njemu naziva ***Moravecov paradoks***. Najkraće kazano sastoji se u tome da rasuđivanje na visokoj razini zahtijeva malo računanja, a računanje na niskoj razini razumijevanja senzorskih poticaja i djelovanja na izvršne uređaje (aktuatore) zahtijeva velike računalne resurse. Moravec je u jednom od svojih radova (Moravec, 1988.) napisao: *„relativno je lako napraviti računala koja pokazuju performanse na razini odraslih osoba na testovima inteligencije ili igranju dame, ali je teško ili gotovo nemoguće dati im vještine jednogodišnjaka kada se radi o percepciji i mobilnosti”*. Ovo je pokrenulo potpuno novi pravac u umjetnoj inteligenciji, ***umjetnu inteligenciju temeljenu na ponašanju***, koju smo već spominjali, a posebno u istraživanjima ***Rodneya Brooksa*** (1954. –) i njegove ***supsumcijske*** arhitekture. Ovaj se smjer umjetne inteligencije zove ***Nouvelle AI***, a kako je ***Brooks*** kazao (Brooks, 2002.): *„Nema spoznaje. Samo osjećanje i djelovanje.”* (engl. *„No cognition. Just sensing and action.”*)

Koncem osamdesetih godina prošloga stoljeća vraća se interes za neuronske mreže te počinje ***faza povratka umjetnih neuronskih mreža*** koja isto tako traje do danas. Umjetne neuronske mreže intenzivno su se izučavale pedesetih godina 20. stoljeća. Prvi put ih spominju 1943. godine neuroznanstvenici ***Warren S. McCulloch*** i logičar ***Walter Pitts***, a popularnost im raste nakon što je 1949. godine ***Donald Hebb*** (1904. – 1985.) postavio temeljni zakon učenja umjetnog neurona poznat kao ***Hebbovo pravilo***. Međutim, umjetne neuronske mreže se koncem 1980-ih godina ne vraćaju samo zbog

toga što ih sada računala mogu bolje pratiti (pa je sve više komercijalnih primjena neuronskih mreža) već i zbog toga što ***Rumelhart*** i ***McClelland*** 1986. godine postavljaju tzv. ***konekcionistički*** *(*engl. *Connectionist)* pristup inteligentnom sustavu koji se temelji na modelu memorije zasnovanom na umjetnoj neuronskoj mreži. Ovo je bitno drugačiji pristup od pristupa inteligenciji simboličkim modelima koji su zastupali ***Nowell*** i ***Simon*** ili logičkom pristupu inteligenciji ***McCarthya***. Početkom 21. stoljeća bilježi se povratak intenzivnih istraživanja umjetnih neuronskih mreža, posebno pojavom tzv. ***dubokih neuronskih mreža*** (engl. *Deep Neural Networks*) (slika 1-12) najčešće vezanih uz strojno učenje i poznatih po nazivu ***duboko učenje*** (engl. *Deep Learning*). Iza 2010. godine imamo industrijsku primjenu dubokog učenja u zadacima prepoznavanja govora. Google 2010. godine uvodi ***Google Brain*** koji je 2012. godine prepoznao sliku mačke, a 2014. godine opisao scenu na slici. 2012. godine pojavljuje se i ***TensorFlow*** programska biblioteka otvorenog koda pogodna za strojno učenje.



Slika 1-12. Duboka neuronska mreža u zadatku prepoznavanja lica

To je i doba kada se razvijaju i novi pristupi pojedinim zadacima koje su postavili istraživači umjetne inteligencije na samim počecima, pa počinje ***faza znanosti umjetne inteligencije***. U području razumijevanja govora intenzivno se razvijaju ***skriveni Markovljevi modeli*** (engl. *HMM – Hidden Markov Models*), a vraća se značaj i strožih matematičkih pristupa postupcima umjetne inteligencije, najviše teorije vjerojatnosti kroz širu primjenu ***Bayesovih mreža*** (engl. *Bayesian Networks*). Posebno je uspješan bio razvoj igara dva igrača. IBM-ov ***DeepBlue*** je 1997. godine pobijedio svjetskog šahovskog šampiona ***Garryja Kasparova***, a 2016. godine Googleov program ***AlphaGO*** pobijedio je korejskog igrača Go-a ***Leeja Sedola***, u to vrijeme najboljeg svjetskog igrača Go-a s rezultatom 4 : 1 (više detalja iz poglavlja 3). IBM-ov program za pronalaženje odgovora na pitanja iz domene općeg znanja ***IBM Watson*** 2011.g. pobjeđuje dva suparnika na kvizu ***Jeopardy!*** (više detalja iza poglavlja Predgovor).

I na kraju kao posljednja faza ističe se ***faza inteligentnih agenata*** koja započinje negdje koncem devedesetih godina prošloga stoljeća i traje do danas. Definira se samostalni programski ili sklopovski entitet nazvan ***agent*** koji samostalno djeluje u svom okružju koristeći pri tome različite postupke umjetne inteligencije. Istraživanja se okreću i prema međudjelovanjima inteligentnih entiteta, pa se intenzivno izučava problematika ***kooperacije***, ***koordinacije*** i ***konkurencije***. Formiraju se inteligentni sustavi temeljeni na agentskoj tehnologiji koji sve češće nalaze primjenu i u komercijalnim sustavima, od koordinacije računala povezanih u GRID arhitekturu, pa do inteligentnog sustava za rano otkrivanje šumskog požara koji se od 2003. godina razvija na FESB-u (vidi sliku 2-5).

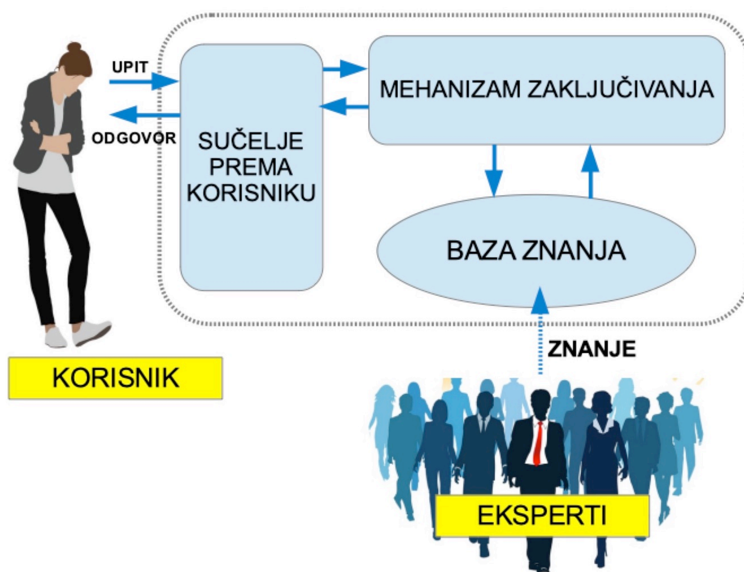
1.6 Rješavanje zadataka postupcima umjetne inteligencije

Umjetna inteligencija koristi se kod rješavanja zadataka koje nije moguće riješiti na uobičajeni, algoritamski način. Algoritamski način rješavanja zadataka je izvođenje manje ili više složenih formula koje na lijevoj strani ima nepoznanicu, a na desnoj poznate veličine povezane odgovarajućim matematičkim operatorima, pa se rezultat dobije direktnim proračunom. Uobičajeno je ovakve metode

nazivati ***jake metode*** (engl. *Hard Methods*). Zadaci koje rješava umjetna inteligencija su najčešće problemski zadaci koji se trebaju postaviti i prikazati na specifičan način, pogodan za rješavanje nekim od postupaka umjetne inteligencije. Tipični postupci rješavanja zadataka koje umjetna inteligencija koristi uključuju ***pretraživanje*** (engl. *Search*) i obično se zovu ***slabe metode*** (engl. *Weak Methods*) kako bi se naglasila razlika između njih i uobičajenih algoritamskih metode klasičnih računalnih programa. Neke od slabih metoda koje koristi umjetna inteligencija su:

- ***Generiraj-i-testiraj*** (engl. *Generate-and-test*) – moguća rješenja pretpostave se bez nekog pravila i razloga, pa se nakon toga svako od njih testira kako bi se ustanovilo je li odgovara rješenju. Ovo je sigurno najprimitivniji postupak, ali kako ćemo kasnije vidjeti teorijski je dosta zanimljiv.
- ***Heurističko pretraživanje*** (engl. *Heuristic Search*) – serijom operacija nastoji se konstruirati put od početne situacije do traženog rješenja koristeći pravila koja se ne mogu dokazati.
- ***Optimalno pretraživanje*** (engl. *Optimal Search*) – pri odabiru sljedećeg koraka koristi se mjera koja nam kaže koliko smo napredovali prema konačnom rješenju.
- ***Planiranje apstrakcijom*** (engl. *Planning-by-abstraction*) – zadatak se najprije pojednostavi, prebaci u veću razinu apstrakcije, pa riješi na toj razini. Nakon toga se takvo pojednostavljeno rješenje koristi kao vodič u rješavanju osnovnog problema.
- ***Poklapanje*** (engl. *Matching*) – sadašnje stanje preslikava se na željeno stanje kako bi se ustanovila njihova korespondencija i u kojem smjeru se treba tražiti put od početnog rješenja prema konačnom.

Da bi riješio postavljeni problemski zadatak metodama umjetne inteligencije, sustav umjetne inteligencije treba imati dio koji nazivamo ***mehanizam zaključivanja*** (engl. *Inference Engine*) i pohranjeno znanje u strukturi koju nazivamo ***baza znanja*** (engl. *Knowledge Base*). Pohranjeno znanje posebnim je postupcima prikupljeno od stručnjaka ili eksperta koji je to znanje imao, formalno prikazano na način da ga računalo može razumjeti i organizirano pohranjeno u bazu znanja kako bi mehanizam zaključivanja do njega lako došao. Kako sve to nekome treba i kako sve to netko koristi, nezaobilazni dio je i ***sučelje prema korisniku*** (engl. *User Interface*). Iako korisnik nema stručno znanje za rješavanje određenog problemskog zadatka, on je sada u mogućnosti doći do rješenja na temelju znanja eksperta koje je pohranjeno u bazu znanja. Strukturu sustava prikazuje slika 1-13.



Slika 1-13. Ilustracija postupka rješavanja zadataka metodama umjetne inteligencije. Korisnik (nestručnjak) želi riješiti zadatak koji preko sučelja prenese mehanizmu za zaključivanje. Njegova je uloga riješiti zadatak na temelju pohranjenog znanja eksperata (stručnjaka) koji ovakav zadatak znaju riješiti, te rješenje vratiti korisniku

Pri tome ne zaboravimo da je kod sustava klasične umjetne inteligencije znanje u bazi znanja stalno prisutno. Osnovna značajka postupka rješavanja zadatka metodama umjetne inteligencije je ***traženje rješenja u prostoru parcijalnih rješenja*** koji nazivamo ***prostor pretraživanja*** (engl. *Search Space*) ili ***prostor rješenja*** ili ***problemski prostor***. Traženje je kombinacijsko i pod nadzorom ga drži znanje koje kaže kako izabrati ili suziti opcije pri svakom koraku traženja. Zbog toga je jedan od osnovnih problema umjetne inteligencije formalno predstavljanje ***znanja*** na način prilagođen pohrani u memoriji računala. Pri tome se obično koristi matematička logika, od klasične predikatne logike, pa do različitih nestandardnih logičkih sustava kao što su temporalne ili neizrazite logike.

Strukturalno, znanje se može prezentirati i pohraniti u obliku različitih ***struktura znanja***. Tipični su primjeri semantičke mreže, okviri, produkcijska pravila. Znanje se gradi na temelju informacija, a postupak koji informacije pretvara u znanje naziva se ***učenje***.

Postoje dvije osnovne kategorije znanja:

- znanje o nečemu ili znanje da nešto jest, i
- znanje o znanju ili znanje o tome kao nešto napraviti, učiniti ili kako postupiti.

U prvu kategoriju znanja spadaju primjerice ***tvrdnje,*** dok u drugu spajaju ***procedure*** i ***planovi,*** pa se takvo znanje o znanju, ili znanje o postupanju znanjem obično naziva ***metaznanje.***

Znanje može biti ***javno*** i ***privatno***. Javno znanje je opće poznato, dostupno u javnim dokumentima i knjigama, dok je privatno znanje osobno, najčešće heuristički temeljeno na vlastitom iskustvu i intuiciji.

Zadaci prikupljanja, prikazivanja i korištenja znanja su u osnovi zadaci istraživanja povezani s ekspertnim (stručnim) sustavima, područjem umjetne inteligencije koje je možda u današnjem stadiju razvoja umjetne inteligencije dosta zanimljivo s komercijalne točke gledišta, pa ćemo se baviti i njima, posebno vezano uz postupke zaključivanja.

1.7 Kriteriji uspjeha

Jedno od najvažnijih pitanja bilo kojeg znanstvenog ili istraživačkog projekta je

„Kako ću znati jesam li uspio ili ipak nisam uspio?”.

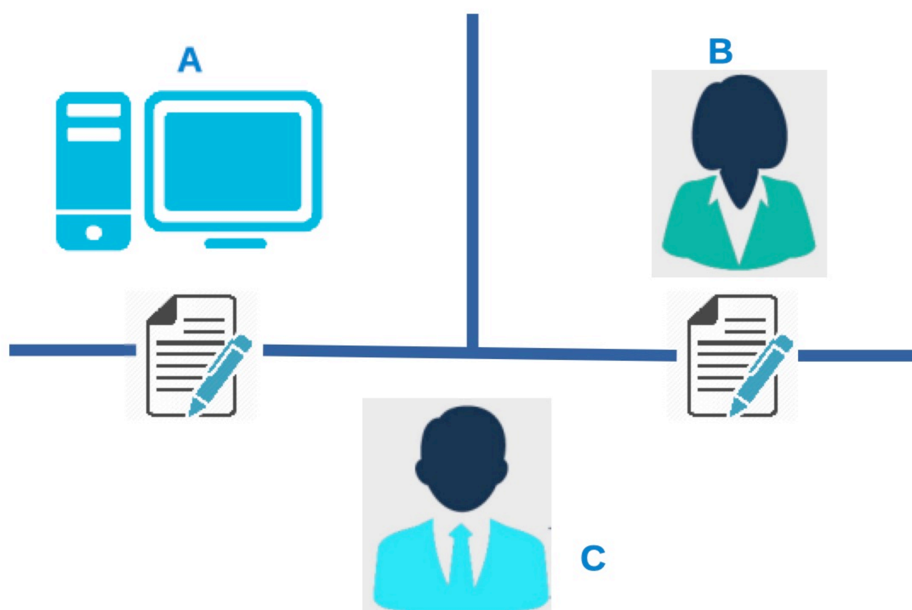
Umjetna inteligencija u tome nije izuzetak.

„Kako ćemo znati jesmo li napravili inteligentnu mašinu?”

Ovo je pitanje jednako teško kao i pitanje *„Što je inteligencija?"* i njime se bavilo dosta teoretičara i praktičara umjetne inteligencije. Još 1950. godine ***Alan Turing*** (1912. – 1954.), jedan od pionira istraživanja umjetne inteligencije, odnosno točnije kazano problematike izgradnje inteligentnog stroja, predložilo je postupak kojim se testira inteligencija stroja. Metoda je poznata pod nazivom ***Turingov test*** (slika 1-14).

Za provođenje testa trebamo dvoje ljudi i stroj koji testiramo. Jedan od ljudi igra ulogu ***ispitivača*** i nalazi se u posebnoj sobi (prednji plan), te s drugim čovjekom i strojem komunicira preko tastature. Ispitivač postavlja pitanja, ali ne zna tko daje odgovor: stroj ili drugi čovjek. Cilj testa je da ispitivač pokuša odgonetnuti koji odgovor daje osoba, a koji stroj. Cilj stroja je odgovorima zbuniti ispitivača kako bi on vjerovao da se radi o ljudskoj osobi, a ne stroju. Ako u tome uspije, onda se za takav stroj može kazati da se radi o stroju koji misli. Na primjer, ako ispitivač postavi pitanje:

„Koliko je 12324 pomnoženo sa 73981?” – stroj može čekati nekoliko minuta i onda dati pogrešan odgovor kako bi zbunio ispitivača s obzirom na to da običan čovjek teško u glavi može jednostavno pomnožiti dva peteroznamenkasta broja.



Slika 1-14. Ilustracija Turingovog testa. A je računalo koje testiramo, C je ispitivač, a B je ljudski odgovaratelj. Računalo A je prošlo Turingov test ako ispitivač C ne može ustanoviti tko daje odgovore – stroj ili čovjek.

Turing je test postavio 1950. godine, a danas još uvijek ne postoji program koji bi ga prošao u općem obliku. Postoje specijalizirani programi koji imaju domensko znanje u određenom području i vrlo vješto programiran postupak konverzacije, ali ako se ispitivač makne iz tog područja, vrlo je lako ustanoviti da se radi o stroju. Od 1990. do 2020. godine održavalo se natjecanje nazvano ***The Loebner Prize***[11] (slika 1-15) kojim su se nagrađivali programi koji su se najviše približili prolasku Turingovog testa.



Slika 1-15. Loebnerova medalja koja se od 1990. godine dodjeljuje programu koji se najviše približi prolasku Turingovog testa[12]

[11] https://en.wikipedia.org/wiki/Loebner_Prize

[12] Izvor slike http://www.paulmckevitt.com/loebner2013/ .

Generalnu nagradu u iznosu od 100 000 američkih dolara nitko nije osvojio, ali se svake godine davala godišnja nagrada u iznosu od 2000 američkih dolara za program koji se najviše razvio u protekloj godini.

Nedostatak Turingovog testa je što zahtijeva verbalnu komunikaciju prirodnim jezikom i što je podložan tzv. ELIZA efektu. ELIZA efekt je primijećen još 1970-ih godina kada je napravljen konverzacijski program ELIZA koji je trebao simulirati psihoterapeuta u duhu psihoterapije psihologa ***Carla Rogersa*** nazvane ***PCT*** – *Person-Centered Therapy*. ELIZA je preko konzole razgovarala s čovjekom te postavljala zahtjeve tipa: *„Kaži mi više o svojoj obitelji"*, ako bi čovjek spomenuo oca, majku, brata ili nekoga iz obitelji. Ovakvi su upiti stvarali kod čovjeka koji je razgovarao s ELIZOM podsvjesni privid da se računalo stvarno ponaša kao čovjek i da ELIZA pokazuje interes za ono o čemu se govori, iako je svjesno znanje govorilo da ona to ne može pokazivati zato što nije programirana da simulira osjećaje. Više detalja o ELIZI u dodatku na kraju ovog poglavlja. Kod pisanja programa koji su se podvrgavali Turingovom testu programeri su namjerno koristili ELIZA efekt, pa umjesto da nastoje povećati stvarno znanje ugrađeno u program, koristili su konverzacijske doskočice kojima bi nastojali zavarati ispitivača.

Zbog toga je 1996. godine Kanađanin ***Kennet Christopher McKinstry*** pokrenuo projekt ***MISTIC*** – *Minimum Intelligent Signal Test Item Corpus*[13] s idejom definiranja tzv. ***MIST*** testa – *Minimum Intelligence Signal Test* kod kojega odgovori na postavljeno pitanje mogu biti samo binarnog oblika (da/ne ili istinito/lažno). Ideja je testa bila definirati kvantitativnu statističku mjeru strojne inteligencije ili kako ju je ***McKinstry*** nazvao ***statističku kvantitativnu mjeru sličnosti čovjeku*** (engl. *Quantitative Statistical Measure of Humanness*). Definirao je 80.000 tvrdnji oblika:

Je li Zemlja planet?

Je li Sunce veće od stopala?

Je li ljudi ponekad lažu?

koje je nazvao ***„mindpixels"*** što bismo mogli slobodno prevesti kao ***pikseli uma***. Stroj odgovara na slučajno odabrana pitanja te se na temelju usporedbe točnih odgovora i vjerojatnosti točnih odgovora uz slučajno odgovaranje na pitanja na neki način mjeri stupanj razumijevanja i inteligencije. Na primjer, stroju se postavi 20 pitanja i on točno odgovori na svih 20 pitanja, a vjerojatnost točnog odgovora na svih 20 pitanja je $1/2^{20} = 1/1\,048\,576$. Ako se test ponovi nekoliko puta, svaki put s novoodabranim pitanjima koja nije poznavao prije odgovaranja, njegova bi se inteligencija mogla čak i kvantitativno izraziti. McKinstry je 2000. godine pokrenuo projekt formiranja što veće baze *mindpixela* te je do 2004. godine uspio skupiti preko 1,4 milijuna tvrdnji. Međutim projekt se prekida 2005. godine, a tragičnom smrću McKinstryja 2006. godine postala je upitna i budućnost prikupljene baze *mindpixela*. U međuvremenu je pokrenuto nekoliko sličnih projekata od kojih posebno ističemo ***Open Mind Common Sense Project (OMCS)***[14] pokrenut 1999. godine na MIT-u kao *open source* projekt koji je do danas prikupio preko milijun činjenica i tvrdnji iskazanih rečenicama prirodnog engleskog jezika uz sudjelovanje više od 15.000 ljudi. OMCS projekt kasnije je prerastao u ***ConceptNet***[15] (logo na slici 1-16). ***ConceptNet*** je slobodno dostupna semantička mrežu projektirana na način da računalima omogući shvaćanje riječi koje koriste ljudi.



Slika 1-16. Logo projekta ConceptNet – slobodno dostupne semantičke mreže s riječima prirodnog jezika

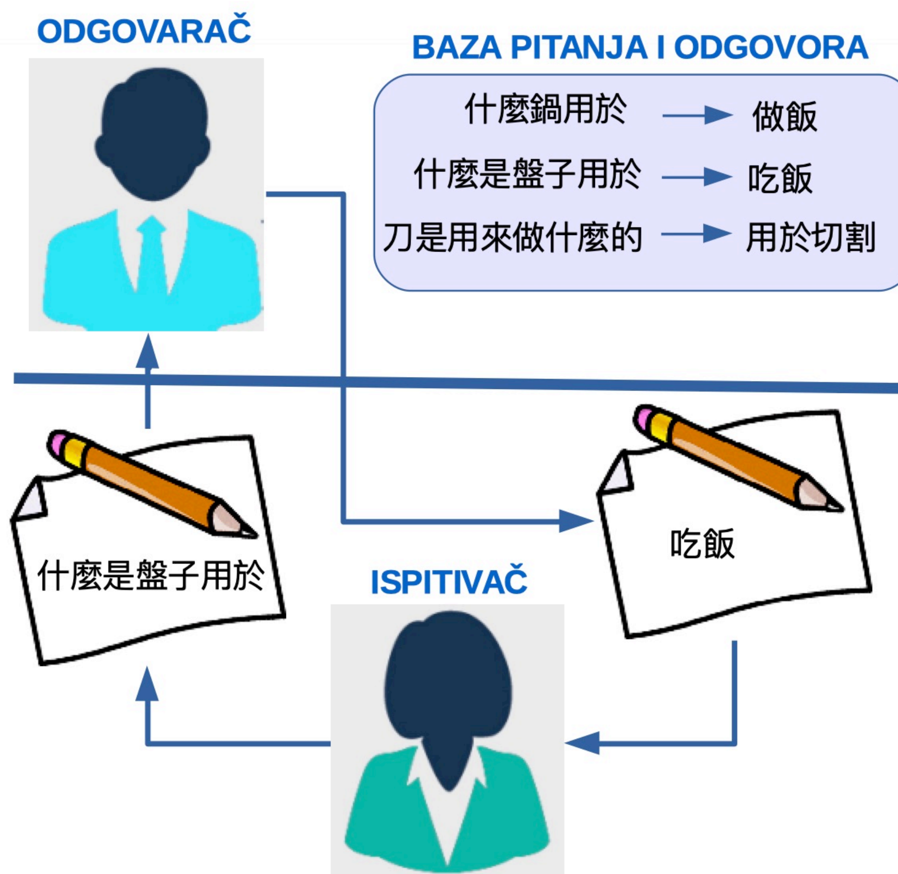
Spomenimo i ***Cyc*** vezan sa zdravorazumskim znanjem koji smo spomenuli na kraju Predgovora.

[13] https://en.wikipedia.org/wiki/Mindpixel

[14] https://en.wikipedia.org/wiki/Open_Mind_Common_Sense

[15] http://openmind.media.mit.edu

Problematikom umjetne inteligencije i njenog testiranja bavili su se brojni teoretičari i filozofi. Neki od njih smatraju da stroj u biti nikada neće biti u mogućnosti razumjeti to što radi, a s tim da nikada neće moći pokazati stvarna svojstva inteligencije. Često se spominje tzv. ***zamjerka kineske sobe*** (engl. *Chinese Room Objection*) ilustrirane na slici 1-17 koju je postavio filozof ***John Searle*** (1932. –) i u kojoj tvrdi da za računalni program nikada nećemo moći kazati da razumije i misli.



Slika 1-17. Zamjerka kineske sobe filozofa Johna Searlea – ispitivač postavi pitanje na kineskom, a odgovarač daje odgovor iako nema pojma kineski i ne razumije što ga se pita, već samo traži poklapanje postavljenog pitanja s nekim od pitanja-odgovora iz dovoljno velike baze pitanja-odgovora

Dokaz se temelji na pretpostavci da se rad računala može usporediti s radom čovjeka koji koristi papir i olovku. Na primjer, osnovna operacija računala je usporedba binarnih nizova smještenih u dvije memorijske lokacije i zapisivanje jedinice u neku treću lokaciju ako se nizovi poklapaju. To isto može raditi i čovjek s papirom i olovkom tako da na papir zapisuje binarne nizove, uspoređuje ih i na novi papir napiše 1 ako se poklapaju. Čovjek to radi na discipliniran, ali neinteligentan način. Zapravo, ako se čovjeku daju detaljna upute za nešto što treba učiniti, on to može učiniti i bez da razumije što točno radi. ***Searle*** to ilustrira Turingovim testom koji se provodi korištenjem kineskog jezika, s tim da se u drugoj prostoriji nalazi čovjek koji se zove Clerk i koji uopće ne zna kineski, ali ima vrlo precizne instrukcije što napraviti s kineskim zapisom koji mu ispitivač gurne kroz vrata u sobu. On ga uzme i po instrukcijama uspoređuje, te na kraju formira novi kineski zapis koji vrati ispitivaču, a da pri tome nije niti razumio o čemu se radi, a još manje smislio odgovor.

Searle proširuje priču na način da se u sobi umjesto Clerka nalazi cijela ekipa ljudi koji su specijalizirani pa svaki obrađuje jedan dio zapisa koji je ispitivač poslao, ali isto tako ni svi kao cjelina ne razumiju što u biti rade, a još manje misle. Samo disciplinirano uspoređuju nizove i zapisuju jedinice ako se oni poklapaju. Sa ***Searlovim*** razmišljanjem neki se slažu, neki ne, ali je činjenica da će proći još vremena prije negoli će neki stroj proći globalni Turingov test.

U užim domenama istraživanja postavlja se pitanje može li se uspjeh umjetne inteligencije mjeriti unutar tih domena. Odgovor je potvrdan. Tipičan je primjer uspješnost programa za igranje šaha koji može dobiti i *„rejting"* u odnosu na *„rejting"* igrača kojeg je pobijedio. Ili na primjer slučaj sa stručnim sustavom DENDRAL koji smo spominjali. Teško je numerički mjeriti uspjeh DENDRAL-a u usporedbi s čovjekom kemičarom, ali podatak da su neke analize DENDRAL-a publicirane kao originalni rezultati istraživanja govori da ga možemo smatrati uspješnim u domeni u kojoj on djeluje. Uspješnost inteligentnih sustava za prepoznavanje šumskog požara može se mjeriti postotkom prepoznatih, postotkom neprepoznatih situacija šumskog požara u nastajanju, te postotkom lažnih alarma. U svakoj domeni moguće je definirati određenu mjeru kojom možemo vrednovati uspješnosti pojedinih pristupa.

1.8 Cilj umjetne inteligencije (umjesto zaključka)

Smatramo li pod inteligentnom aktivnošću stroja postupke kojima se stroj približava inteligentnom djelovanju čovjeka, postavlja se pitanje do koje mjere stroj može zamijeniti čovjeka. Danas smo na razini kod koje se razmišlja da je uloga inteligentnog stroja da pomaže čovjeku kod

- obavljanja rutinskih, jednostavnih i ponovljivih zadataka
- inteligentnog asistiranja i savjetodavnog djelovanja i
- da ga eventualno zamijeni u specifičnim aktivnostima.

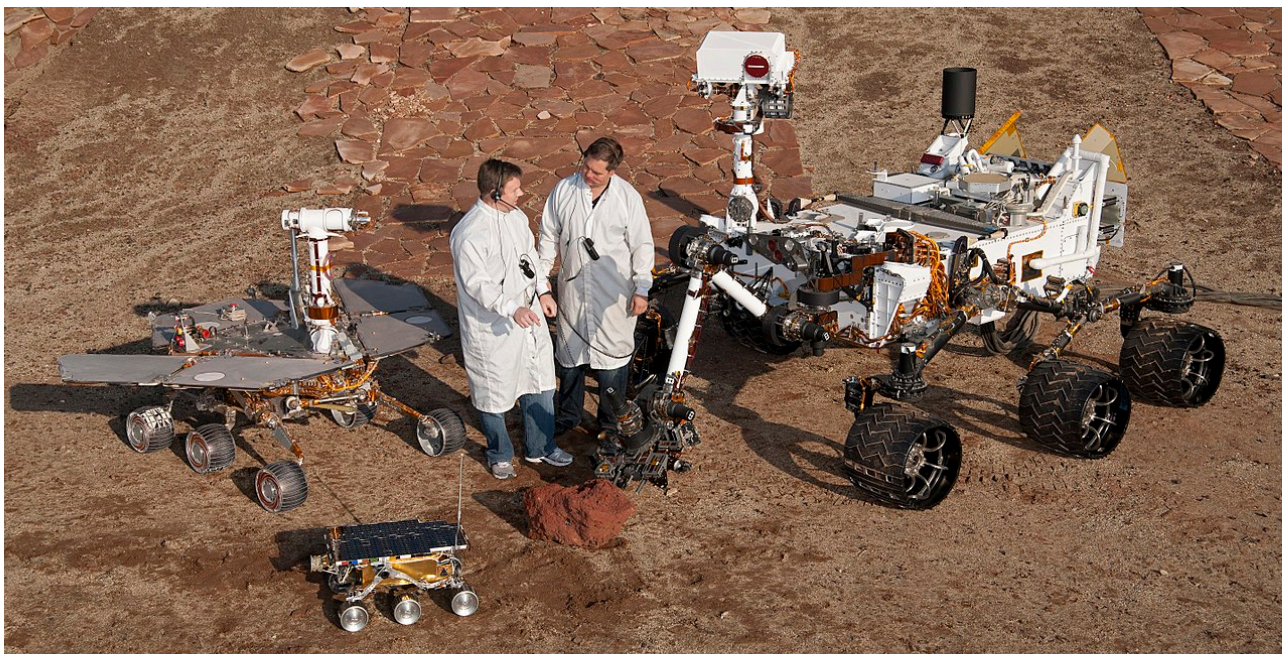
Prvu grupu zadataka mogu obavljati postojeća računala, a korištenjem specijalnih jezika umjetne inteligencije (npr. Lisp) moguće je postići i zavidni stupanj složenosti ovakvih zadataka.

Druga grupa zadataka također je u središtu današnjeg interesa i o računalu se govori kao o sredstvu kojim se povećavaju ljudske sposobnosti. Računala služe kao *„pojačala znanja"*. Cilj nam je inteligentnim sustavom povećati sposobnosti pojedinca i kvalitetu njegovog djelovanja dajući mu na raspolaganje i sposobnosti cijelog niza stručnjaka ili eksperata. Ovo je područje stručnih sustava, možda danas jedno od najzanimljivija s komercijalne strane gledišta. Uobičajeni postupci pri razvoju sustava sa znanjem stručnjaka ili eksperata uključuje nekoliko grupa ljudi s različitim tipovima obuke. Prije svega treba nam hardverski i softverski inženjer koji je spona između stručnjaka koji najčešće nema nikakvo ili slabo znanje o računalima i samog računala u kojem će to znanje biti pohranjeno. Kako stručnjak često ne može precizno objasniti razloge svojih postupaka, jednostavne metode prenošenja njegovog znanja programeru – softverskom inženjeru pokazale su se neadekvatnim pa je došlo do potrebe uvođenja potpuno nove struke ljudi, tzv. ***inženjera znanja***, koji imaju i vrlo specifično obrazovanje nastalo kombinacijom znanja iz računarstva, ali i znanja sociologije, eksperimentalne i kognitivne psihologije. Zadatak inženjera znanja je na odgovarajući način izvući znanje iz stručnjaka, pripremiti ga u neki od formalnih oblika za pohranu znanja te primijeniti pri samostalnom djelovanju stručnog sustava. U okviru ovog udžbenika mi se prvenstveno bavimo samo tim matematičko-tehničkim dijelom – pohranom i primjenom znanja.

O *trećoj grupi zadataka* do sada se razmišljalo prvenstveno u situacijama koje su izuzetno opasne za čovjeka. Primjeri su inteligentni samohodni robot za otkrivanje i neutraliziranje eksploziva, robot za inspekciju dijelova nuklearne elektrane u kojoj je povećan stupanj radijacije, ili najpoznatiji od svih ***Mars Rover*** – samohodno vozilo – robot za istraživanje površine planeta Mars (slika 1-18), ali se u posljednjih par godina javljaju i primjeri samostalnih robotskih vozila koji ne trebaju vozača ili samostalnih robotskih brodova koji sami, bez posade, prevoze teret preko oceana, a kapetan sjedi u virtualnoj upravljačkoj sobi na kopnu.

Ovakvih će primjera biti sve više pa se otvaraju i etička pitanja suživota ljudi i sustava umjetne inteligencije. Ovo je prepoznala i Europska unija, pa je 8. travnja 2019. godine prezentirala dokument ***„Etičke smjernice za pouzdanu umjetnu inteligenciju"*** (engl. *Ethics guidelines for trustworthy AI*)[16] (slika 1-19).

[16] https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai



Slika 1-18. Robotska vozila projekta Mars rover. Naprijed lijevo je Sojourner iz 1997. godine (dug 65 cm), iza lijevo je Mars Exploration Rovers (MER) iz 2004. (dug 1,6 m – kopija vozila Spirit i Opportunity), a desno je Curiosity iz 2012. godine (dug 3 m)[17] U ožujku 2021.g. na Mars je sletio novi rover Perseverance, iste dužine kao i Curiosity, ali nešto teži, koji ujedno nosi i mali robotski helikopter Ingenuity[18]

Dokument je pripremila Stručna skupina na visokoj razini o umjetnoj inteligenciji (AI-HLEG) koju je sačinjavao 51 eminentni stručnjak s različitih sveučilišta, instituta i tvrtki. Nastao je na temelju prijedloga objavljenog u prosincu 2018. godine i više od 500 komentara koji su primljeni tijekom procesa otvorene konzultacije. Prema dokumentu (citat iz hrvatske verzije dokumenta sa str. 6, stavke 15 i 16 [19]):

*„(15) Pouzdana umjetna inteligencija ima **tri sastavnice**, koje trebaju biti ispunjene tijekom cijelog životnog ciklusa sustava:*

*trebala bi biti **zakonita** i osigurati poštovanje svih primjenjivih zakona i propisa;*

*trebala bi biti **etična** i osigurati poštovanje etičkih načela i vrijednosti; i*

*trebala bi biti **otporna** i iz tehničke i iz socijalne perspektive jer sustavi umjetne inteligencije, čak i s dobrim namjerama, mogu uzrokovati nenamjernu štetu.*

(16) Svaka je sastavnica nužna, ali sama po sebi nije dovoljna za postizanje pouzdane umjetne inteligencije. U idealnom su slučaju sve tri sastavnice usklađene i preklapaju se u svojem djelovanju. Međutim, u praksi mogu postojati napetosti među tim elementima (npr. ponekad područje primjene i sadržaj postojećeg zakona nisu usklađeni s etičkim normama). Naša pojedinačna i zajednička odgovornost kao društva jest surađivati kako bismo osigurali da sve tri sastavnice pomognu u postizanju pouzdane umjetne inteligencije."

[17] Slika je s https://en.wikipedia.org/wiki/Curiosity_(rover) uz Public Domain licencu.

[18] Više detalja na https://mars.nasa.gov/mars2020/

[19] https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=60428



Slika 1-19. Preporuke EU-a za projektiranje pouzdanih sustava umjetne inteligencije

Dokument vrlo detaljno opisuje što se smatra pod ***zakonitom*** (engl. *Lawful*), ***etičnom*** (engl. *Ethical)* i ***otpornom*** (engl. *Robust*) umjetnom inteligencijom i nastoji pružiti općenite upute za primjenu umjetne inteligencije. Smjernice su prikazane u tri razine apstrakcije, od prve najapstraktnije koje daju ***temelje pouzdane umjetne inteligencije*** koja treba osiguravati:

- zadovoljavanje poštovanja ljudske autonomije
- sprječavanje nastanka štete
- pravednost i
- objašnjivost,

preko druge razine koja razmatra ostvarenje pouzdane umjetne inteligencije kroz:

- ljudsko djelovanje i nadzor
- tehničku otpornost i sigurnost
- privatnost i upravljanje podacima
- transparentnost
- raznolikost, nediskriminacija i pravednost
- dobrobit društva i okoliša i
- odgovornost,

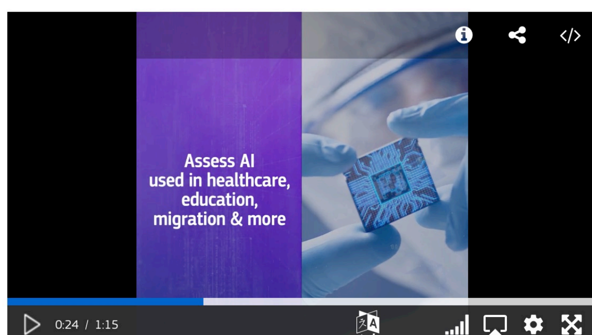
do treće, najkonkretnije razine koja se bavi ***procjenom pouzdane umjetne inteligencije***.

Smjernice imaju ukupno 48 stranica i trebale bi biti obvezno štivo za sve koji se bave razvojem i primjenom umjetne inteligencije, a takvih će u budućnosti biti sve više. Sustavi umjetne inteligencije djeluju tu oko nas, sudjeluju u našem svakodnevnom životu i sigurno će tu i ostati.

Nakon ovog dokumenta slijedio je i niz drugih dokumenata koji su na kraju objedinjeni u platformu: Povjerenje i izvrsnost za umjetnu inteligenciju[20] (slika 1-20) kojoj je cilj promocija istraživanja umjetne inteligencije na EU razini u skladu s etičkim smjernicama za pouzdanu umjetnu inteligenciju. Doneseni su i brojni dokumenti i planovi koji bi trebali u sljedećem razdoblju EU dovesti u novo digitalno doba, jednim dijelom temeljeno i na umjetnoj inteligenciji. EU će do 2030. g. godišnje ulagati 1 milijardu € u istraživanje i primjenu pouzdane umjetne inteligencije u okviru programa Digitalna Europa i Obzor Europe, te privući više od 20 milijardi € ulaganja kroz Mehanizam za oporavak i otpornost.

Povjerenje i izvrsnost za umjetnu inteligenciju

Koristi od pouzdane umjetne inteligencije mogu biti višestruke, na primjer bolja zdravstvena zaštita, sigurniji i čišći promet, učinkovitija proizvodnja te jeftinija i održivija energija.



Koristi od pouzdane umjetne inteligencije mogu biti višestruke, na primjer bolja zdravstvena zaštita, sigurniji i čišći promet, učinkovitija proizvodnja te jeftinija i održivija energija. EU-ov pristup umjetnoj inteligenciji ohrabrit će ljude da prihvate te tehnologije, a poduzeća potaknuti da ih razvijaju.

SADRŽAJ STRANICE

EU i umjetna inteligencija

Izgradnja povjerenja prvim pravnim okvirom o umjetnoj inteligenciji

Jačanje izvrsnosti za UI

Projekti u području UI-ja koje financira EU

Koristi umjetne inteligencije

EU i umjetna inteligencija



Umjetna inteligencija (UI) može pomoći u rješavanju mnogih problema u društvu. No to je moguće samo ako je tehnologija visokokvalitetna i ako se njezinim razvojem i korištenjem stječe povjerenje građana. Zbog toga će strateški okvir EU-a koji počiva na vrijednostima EU-a građanima uliti povjerenje da prihvaćaju rješenja koja se temelje na umjetnoj inteligenciji, a poduzeća potaknuti da ih razvijaju i primjenjuju.

Slika 1-20. Portal EU – Povjerenje i izvrsnost za umjetnu inteligenciju

Naglasimo na kraju i osnovne razlike između tradicionalnih tehnika programiranja i programiranja za potrebe umjetne inteligencije.

Kod ***tradicionalnog programiranja*** programer piše seriju instrukcija u jasnoj liniji od definiranja problema, pa do njegovog rješenja. Svako grananje u slijedu mora biti precizno analizirano i definirano unaprijed. Napredak *„tradicionalne”* računarske znanosti očituje se prvenstveno u brzini proračuna i programskim okružjima koja olakšavaju programiranje i testiranje programa.

[20] https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/excellence-trust-artificial-intelligence_hr#eu-i-umjetna-inteligencija

Umjetna inteligencija uglavnom barata sa simbolima. Simboli predstavljaju veličine iz realnog svijeta, ali, umjesto direktnog računanja kao kod tradicionalnog programa, u ovom slučaju manipulira se simbolima i odnosima među njima. Pri tome se provode manipulacije, a da se u stvari niti ne zna kome će koji simboli biti pridruženi. To se radi u zadnji trenutak nakon pronalaženja funkcionalnog rješenja na razini simbola. Prema tome, kod sustava umjetne inteligencije nije dovoljno samo spremiti podatke ili činjenice kao kod ***baza podataka*** već je potrebno znati i značenje tih činjenica te odnose između pojedinih činjenica. U ovom slučaju govorimo o znanju i njegovoj pohrani u ***bazama znanja***. Ne smijemo zaboraviti da sve to skupa radimo kako bismo mogli rješavati zadatke za koje je potrebno određeno znanje. Zbog toga se najprije bavimo problemom postavljanja i analize postavljenog zadatka, da bismo nakon toga razmatrali kojim postupcima zadatke pokušava riješiti umjetna inteligencija.

....

```
  Welcome to
                    EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                    EE      LL       II       ZZ   AA   AA
                    EEEEE   LL       II      ZZZ   AAAAAAA
                    EE      LL       II     ZZ     AA   AA
                    EEEEEE  LLLLLL  IIII  ZZZZZZ   AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.

ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```



Dodatak: Jedna od prvih praktičnih primjena umjetne inteligencije bio je konverzacijski program temeljen na analizi teksta nazvan ELIZA. Razvijao ga je od 1964.g. do 1966.g. u *MIT Laboratoriju za umjetnu inteligenciju* ***Joseph Weizenbaum*** (1923. – 2008.) na slici desno[21]. ELIZA se temeljila na postupku ***podudaranju uzoraka*** (engl. *Pattern Matching*) dajući iluziju da razumije postavljena pitanja. Temeljila se na ***skriptama*** (engl. *Scripts*) napisanih u programskom jeziku za procesiranje lista SLIP (**S**ymmetric **LI**st **P**rocessor) koji je 1960.g. također razvio Weizenbaum. Jedna od najpoznatijih skripti ELIZE bila je skripta DOCTOR virtualni psihoterapeut prema školi ***Rogerianske psihoterapije*** psihologa ***Karla Rodgersa*** (1902.-1987.). Rogersov pristup terapije usmjerene na pacijenta (engl. ***PCT*** – *Person-Centered Therapy*) u velikoj mjeri temeljio na tome da psihoterapeut pacijentu odgovori koristeći ono što je on sam izgovorio. Weizenbaum je skriptu napravio kao parodiju, a odgovori ELIZE su često zbunjivali korisnike, ostavljajući im dojam da ELIZA stvarno daje inteligentne odgovore.

ELIZA je bila začetnik cijelog jednog područja u umjetnoj inteligenciji vezanog s Turingovim testom. Nakon nje su slijedili brojni konverzacijski programi simulacije razgovora s ljudskim korisnikom prirodnim jezikom koji su se od 1990. do 2020. natjecali za ***Loebnerovu nagradu*** (str.26). Danas su poznatiji pod pojmom *„**chatbot**"*. Chatbotovi su postali sastavni dio brojnih Internet usluga prema korisnicima. Indijska vlada je 2020.g. lansirala chatbot ***MyGov Corona Helpdesk*** koji odgovarao na pitanja o pandemiji Corona virusa. Sastavni su dio i brojnih računalnih platformi (*Apple Siri*, *Amazon Alexa*, *Microsoft Cortana*) a zadatak im je glasovna komunikacija s korisnikom i izvršavanje naredbi zadanih izgovorenom porukom. Predviđa se da će ovakvi sustavi glasovne komunikacije čovjeka korisnika i računala uskoro biti i sastavni dio brojnih uređaja kojima uobičajeno dodajemo naziv *„inteligentni"*, na primjer inteligentni kućanski aparati, inteligentne kuće, inteligentna (autonomna) vozila, itd.

[21] Slike je po CC BY-SA 3.0 licenci sa https://en.wikipedia.org .

2 KOMPLEKSNI ZADACI I NJIHOVO RJEŠAVANJE



”

Kompleksni zadaci su zamršeni, zapleteni i komplicirani, a za njihovo rješavanje ne postoje formule u koje bismo uvrstili poznate vrijednosti i izračunali nepoznate. Kompleksni zadaci rješavaju se na potpuno drugačije načine koristeći postupke kao što su pretraživanje, zaključivanje ili strojno učenje. Bez obzira koja se tehnika rješavanja koristi, prvi je korak postavljanje zadatka, nakon čega slijedi analiziranje zadatka i tek onda rješavanje koje se sastoji u odabiru najprimjerenije tehnike rješavanja zadataka.

Pri izgradnji sustava sposobnog za rješavanje nekog određenog zadatka trebamo voditi računa o tri koraka sustavnog rješenja kojeg čini:

1. ***Postavljanje zadatka*** kod kojeg definiramo prostor stanja u kojem se nalaze moguća rješenja zadatka, od početnog stanja ili situacije u kojoj se trenutno nalazimo, preko svih međustanja do željenog ili ciljnog stanja ili situacije. Početno stanje može biti samo jedno, a ciljnih stanja može biti više.

2. ***Analiziranje zadatka*** kako bismo otkrili sve njegove osobine, te odabrali odgovarajuće metode rješavanja.

3. ***Rješavanje zadatka*** sastoji se u odabiru najbolje (najprimjerenije) tehnike dolaska od početnog stanja do ciljnog stanja koje nam predstavlja rješenje.

2.1 Postavljanje zadataka

Prvi korak pri rješavanju bilo kojeg zadatka, a pogotovo kada se radi o kompleksnom, netrivijalnom zadatku, je postavljanje zadatka, formalni opis onoga gdje se nalazimo, gdje želimo ići, što nam je cilj i kako možemo do tog cilja doći. Nas prije svega zanima postavljanje zadatka na način da ga možemo riješiti nekim od postupaka umjetne inteligencije. To je dosta drugačiji pristup rješavanju zadatka od **standardnog algoritamskog pristupa** s kojim smo se uobičajeno susretali. Algoritamskim pristupom do rješenja se dođe uvrštavanjem poznatih vrijednosti u određenu formulu. Pri tome nas prije svega zanimaju problemski zadaci, pa da bismo uočili razliku između pristupa rješavanju kako se to radi u okviru umjetne inteligencije, pogledajmo najprije kako se problemski zadatak postavlja kod algoritamskog rješavanja. Koristili smo ga tijekom cijelog školovanja, od rješavanja računskih pričica, pa do rješavanja i zadataka iz fizike koji i nisu jednostavni. Tri su osnovna koraka:

- definiranje poznatih vrijednosti

- definiranje nepoznanica, tražene vrijednosti nepoznate veličine koju želimo izračunati i
- definiranje formule, matematičkog izraza koji ćemo koristiti kod rješavanja.

Postupak rješavanja sastoji se od uvrštavanja poznatih vrijednosti u formulu i dobivanja nepoznatog rješenja. Pogledajmo jednostavni primjer:

Zadatak 1. – Standardni algoritamski pristup rješavanju zadataka

Izračunaj brzinu vozila koje je put od 100 m prevalilo za 10 sekundi ako se gibalo jednoliko?

Postavljanje zadatka

1. Poznate vrijednosti:

Put = 100 m

Vrijeme = 10 sekundi

2. Cilj, nepoznata, tražena vrijednost:

Brzina = ?

3. Potrebno znanje – matematička formula:

Brzina = Put / Vrijeme

Rješavanje zadatka – uvrštavanje poznatog u formulu:

Brzina = 100 m / 10 s = 10 m/s

Ponekad zadatak može biti i znatno složeniji pa da bismo došli do rješenja, treba nam i nekoliko formula, ponekad prilično zahtjevnih, ali kod algoritamskog pristupa rješenje uvijek dobijemo direktnim matematičkim proračunom.

Postupci rješavanja zadataka metodama umjetne inteligencije bitno su različiti. Riješiti zadatak znači pronaći put od početnog stanja do ciljnog stanja unutar prostora svih mogućih međustanja. Međustanja predstavljaju moguća rješenja zadatka. Način prelaska iz jednog međustanja u drugo definiraju pravila prelaska. Sastavni dio postupka uključuje u svakom koraku usporedbu trenutnog stanja s ciljnim rješenjem. Ako je postignuto poklapanje, zadatak smo uspješno riješili. Sada je i lakše shvatiti zamjerku kineske sobe. U osnovi se postupci umjetne inteligencije sastoje u generiranju rješenja, usporedbi s ciljnim rješenjem i *„zapisivanju jedinice ako se rješenje i ciljno rješenje poklapaju"*.

Ovakav postupak rješavanja zadataka zahtijeva i drugačije postavljanje zadatka, pri čemu je važan formalni zapis, po mogućnosti u takvom obliku iz kojeg se lako može prebaciti u programski kod. Postavljanje zadataka primjereno postupku umjetne inteligencije sastoji se od četiri koraka:

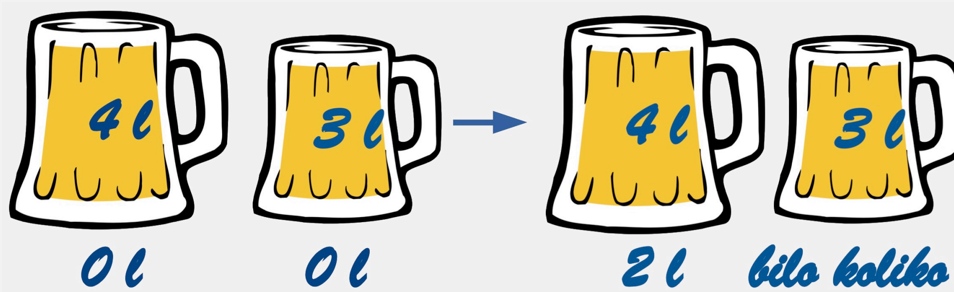
- ***Definiranja prostora stanja***: Potrebno je definirati prostor stanja u kojemu se zadatak rješava. Prostor stanja rješavanja zadatka čine sva moguća rješenja zadatka, a svako se rješenje prikazuje na formalan način. Zbog toga se često naziva i ***prostor rješenja***, ***prostor pretraživanja*** ili ***problemski prostor.***
- ***Definiranje početnog stanja:*** Unutar prostora stanja rješavanja zadatka trebamo odrediti ***početno stanje***, stanje koje smo zatekli, stanje iz kojeg krećemo u rješavanje zadatka.
- ***Definiranja ciljnih stanja:*** Potrebno je definirati stanja koja možemo smatrati prihvatljivim rješenjem i koja nazivamo ***ciljnim stanjima*** ili ***konačnim rješenjima***. Može ih biti i više od jednog. U svakom koraku rješavanja ciljna se stanja uspoređuju s trenutnim stanjem. Njihovo poklapanje znači da smo došli do rješenja.
- ***Definiranja skupa pravila:*** Pravila opisuju moguće akcije ili operacije kojima se iz jednog stanja može doći do nekog drugog stanja unutar prostora rješenja. Ova pravila u biti predstavljaju znanje kojim se vodimo pri traženju rješenja zadatka.

Potrebno je naglasiti da se postupcima umjetne inteligencije problemski zadatak rješava pretraživanjem rješenja u prostoru stanja upotrebom pravila uz pomoć odgovarajuće ***upravljačke***

strategije. Upravljačka strategija određuje u koje ćemo novo stanje otići iz stanja u kojem se trenutno nalazimo. O efikasnosti upravljačke strategije ovisi i efikasnost dolaska do ciljnog rješenja. Prema tome osnovni postupak rješavanja zadataka umjetne inteligencije jest postupak ***pretraživanja*** (engl. *Search*) o kojem detaljno govorimo u poglavlju 3. U nastavku će se na dva primjera ilustrirati način postavljanja i rješavanja zadatka na način koji se koristi u umjetnoj inteligenciji.

Zadatak 2. - Problem punjenja vrčeva (engl. *Water Jug Riddle*)

Imamo dva vrča, jedan od 4 litre (vrč A) i jedan od 3 litre (vrč B). Ni jedan od njih nema oznake o količini vode u vrču i jedina pouzdana informacija o tome može se dobiti samo kada su vrčevi prazni ili puni. Vrčevi se mogu puniti na slavini i prazniti na način da se voda može prelijevati iz vrča u vrč ili izlijevati na pod. Zadatak glasi ovako: ako se krene od situacije praznih vrčeva, kako u vrč A od 4 litre uliti točno 2 litre vode? Budući da na vrčevima ne postoje oznake ne možemo kod prelijevanja vode iz jednog vrča u drugi puniti samo do određene razine. Zadatak bi u tom slučaju bio trivijalan i riješili bismo ga u dva koraka (napunili vrč B te ga izlijevali u vrč A dok u njemu ne bi bilo točno pola tekućine). Vrč možemo puniti ili samo do samog vrha ili svu tekućinu iz jednog vrča preliti u drugi. Zadatak ilustrira slika 2-1.



Slika 2-1. Zadatak dva vrča

Postavljanje zadatka

Trebamo definirati prostor stanja, početno stanje, ciljna stanja i skup pravila:

→ ***prostor stanja*** rješavanja zadatka možemo definirati uređenim parom *(x,y)* gdje *x* može biti 0, 1, 2, 3 ili 4 i predstavlja broj litara vode u vrču A od 4 litre, a y može biti 0, 1, 2 ili 3 i predstavlja broj litara vode u vrču B od 3 litre

→ ***početno stanje*** – oba vrča su prazna, što u prostoru stanja znači (0, 0)

→ ***ciljno stanje*** – zadatak traži 2 litre vode u vrču od A od 4 litre, dok količina vode u vrču B od 3 litre nije važna. To možemo prikazati kao (2, y) gdje je y bilo koja vrijednost iz skupa {0,1,2,3}

→ ***skup pravila*** – početne pretpostavke - vrčevi se mogu puniti i prazniti. Kako ne možemo vidjeti koliko je točno vode ostalo ili koliko je vode doliveno u vrč (nemamo oznaka) količinu vode procjenjujemo samo na temelju pamćenja koliko smo vode izlili. Jedino što možemo vidjeti jest je li vrč pun i tada znamo da je u vrču A 4 litre, a u vrču B 3 litre ili je li neki od vrčeva prazan kada je u njemu 0 litara. Ovo znanje o punjenju – pražnjenju vrčeva trebamo iskazati na formalni, po mogućnosti matematički način kako bismo ga mogli primijeniti prilikom prijelaza iz jednog stanja napunjenosti vrčeva u drugo. Najpogodniji način formalnog zapisa dozvoljenih načina punjenja – pražnjenja je u obliku pravila prijelaza koja matematički prikazujemo relacijom implikacije. Na lijevoj strani implikacije je stanje u kojem se dotično pravilo može primijeniti, a na desnoj je stanje koje će se dobiti nakon primjene tog pravila. Ukupno je 8 dozvoljenih pravila prijelaza:

1. *Napuniti 4-litarski vrč.* $(x, y \mid x < 4) \rightarrow (4, y)$
2. *Napuniti 3-litarski vrč.* $(x, y \mid y < 3) \rightarrow (x, 3)$
3. *Isprazniti 4-litarski vrč na zemlju.* $(x, y \mid x > 0) \rightarrow (0, y)$

4. *Isprazniti 3-litarski vrč na zemlju.* $(x, y \mid y > 0) \rightarrow (x, 0)$
5. *Isprazniti svu vodu iz 4-litarskog vrča u 3-litarski.* $(x, y \mid x + y \leq 3 \ \& \ x > 0) \rightarrow (0, x+y)$
6. *Isprazniti svu vodu iz 3-litarskog vrča u 4-litarski* $(x, y \mid x + y \leq 4 \ \& \ y > 0) \rightarrow (x+y, 0)$
7. *Izliti vodu iz 4-litarskog vrča u 3-litarski dok nije pun.*
   $(x, y \mid x + y \geq 3 \ \& \ x > 0) \rightarrow (x-(3-y), 3)$
8. *Izliti vodu iz 3-litarskog vrča u 4-litarski dok nije pun.*
   $(x, y \mid x + y \geq 4 \ \& \ y > 0) \rightarrow (4, y-(3-x))$

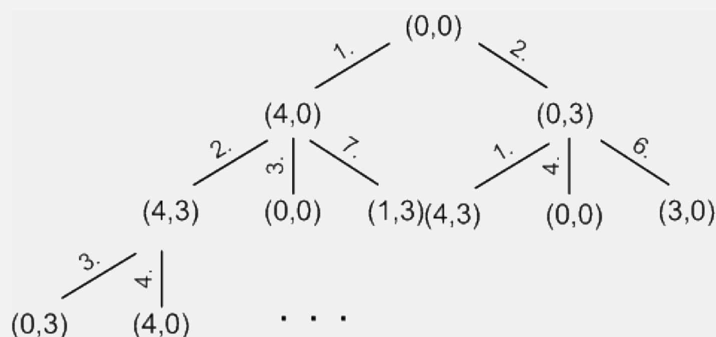
Rješavanje zadatka

Da bi se riješio zadani zadatak osim pravila prijelaza koja predstavljaju znanje o prijelazu iz jednog stanja u drugo, treba nam i upravljačka strategija koja nas vodi kroz pravila. U ovom slučaju možemo koristiti jednostavnu upravljačku strategiju koja kaže:

Usporedi lijevu stranu implikacije s trenutnim stanjem te ako je zadovoljena primjeni to pravilo. Ciklički prolazi kroz sva pravila od prvog do osmog.

Kasnije ćemo vidjeti da odabir ovakve strategije odgovara postupku koji se zove pretraživanje po dubini. Naglasimo da o *efikasnosti strategije upravljanja ovisi i efikasnost postupka rješavanja* postavljenog zadatka.

Strategija treba biti efikasna, a efikasna je prije svega ako *uzrokuje kretanje* ali pri tome kretanje ne smije biti kretanje u krug. Neuspješna strategija bila bi ona koja bi se prolazom kroz nekoliko stanja ponovo vratila u početno stanje, a da ni jedno od prijeđenih stanja nije bilo ciljno stanje. Takva strategija nikada ne bi došla do rješenja. Drugi zahtjev je da strategija bude *sustavna.* Uzmimo za primjer nešto drugačiju strategiju. Na svakom koraku slučajno biramo pravilo koje ćemo testirati. Ova strategija uzrokuje kretanje i možda će dovesti do rješenja, ali to nije sigurno i vjerojatno ćemo napraviti puno nepotrebnih koraka prije nego dođemo do cilja. Strategija u ovom slučaju nije sustavna. Sustavna strategija je izgradnja cijelog prostora mogućih rješenja koji možemo prikazati grafom tipa stablo. Na vrhu, u početnom ili korijenskom čvoru je početno stanje, a ciljno stanje će ako do njega dođemo biti negdje na dnu poslije prolaska kroz određeni broj razina stabla. Slika 2-2 prikazuje nekoliko razina prostora rješenja zadataka dva vrča.



Slika 2-2. Nekoliko razina prostora rješenja zadatka dva vrča

Početno stanje je (0, 0). To se stanje poklapa s lijevom stranom 1. i 2. pravila, pa su dva moguća sljedeća stanja (4, 0) i (0, 3). Primijenimo li pravilo 1. dobijemo stanje (4, 0), dok pravilo 2. daje stanje (0, 3). Svako od ovih stanja uspoređujemo s ciljnim stanjem (2, y). Ako se stanja ne podudaraju nastavljamo dalje graditi prostor rješenja zadatka. Proces se nastavlja sve dok se neko od stanja ne poklopi s ciljnim stanjem. Pri tome se mogu koristiti različite strategije razvoja prostora rješenja o kojima ćemo kasnije više govoriti. Ovdje ćemo naglasiti jedino to da je posebno važno izbjeći petlje, na način da se nastoji spriječiti da se pojedina stanja ponovo pojave. Na primjer stanje (0,0) je početno stanje, ali ono se i dva puta javlja u drugom redu razvoja. Isto tako stanje (4,3) se u istom redu dva puta ponavlja. Efikasna strategija formiranja prostora rješenja je ona koja će ove čvorove isključiti iz daljnjeg razvoja. Zadaci ovakvoga tipa nemaju samo jedno rješenje.

Jednakovrijednih rješenja može biti i više. Na primjer ako rješenje zadatka dvaju vrčeva vrednujemo po broju prijeđenih koraka od početka do cilja, dva moguća jednakovrijedna rješenja su[22]:

$$(0,0)\xrightarrow{1.}(4,0)\xrightarrow{7.}(1,3)\xrightarrow{4.}(1,0)\xrightarrow{3.}(0,1)\xrightarrow{1.}(4,1)\xrightarrow{7.}(2,3)$$

$$(0,0)\xrightarrow{2.}(0,3)\xrightarrow{6.}(3,0)\xrightarrow{2}(3,3)\xrightarrow{8.}(4,2)\xrightarrow{3.}(0,2)\xrightarrow{6.}(2,0)$$

Broj iznad strelice označava koje smo pravilo primijenili. U oba se slučaja do rješenja dolazi nakon šest koraka pa su, što se tiče brzine dolaska od početnog do ciljnog čvora, identični. Međutim postoji razlika u postupku kojim dolazimo do ovog slijeda koraka. Jedno od ovih rješenja može se pronaći brže uz manji razvoj prostora rješenja, pa nam je takvo rješenje pogodnije zato što do njega dolazimo s manje truda.

Kod ovakvog tipa zadatka važno je uočiti da smo prostor rješenja zadatka razvijali tijekom rješavanja zadatka. Spuštali smo razinu po razinu i postupno razvijali prostor rješenja, a u svakom smo koraku koristili bazu znanja i u njoj tražili odgovarajuće pravilo primjenjivo baš za to stanje. Rješavanja zadatka postupkom umjetne inteligencije u ovom se primjeru svodi na sustavno razvijanje prostora rješenja zadatka odgovarajućom strategijom kako bismo pronašli stanje koje odgovara ciljnom stanju.

Zadatak dva vrča je primjer zadatka kod kojeg je relativno jednostavno definirati cilj, pravila i prostor stanja, ali to nije uvijek tako. Postoje zadaci u kojima je teško definirati i prostor rješenja, a ponekad i ciljno stanje i pravila prijelaza. Uzmimo za primjer zadatak razumijevanja rečenice govornog jezika. Ovdje se ni jedna od značajki zadatka ne može jednostavno postaviti.

Sustav kod kojeg je znanje u obliku skupa pravila, strategija upravljanja i baza podataka u koju se spremaju svi podaci bitni za rješavanje zadatka naziva se ***produkcijski sustav***. Velik broj stručnih sustava koji su u upotrebi upravo je oblika produkcijskog sustava, pa ćemo se toj problematici produkcijskog sustava još puno puta vraćati.

Pogledajmo i drugačiji tip problema koji možemo riješiti metodom umjetne inteligencije.

Zadatak 3. - Problem trgovačkog putnika (engl. *Travelling Salesman Problem*)

Trgovački putnik treba na Braču obići četiri mjesta: Supetar, Pučišća, Sumartin i Nerežišća te se vratiti u početni grad. Svaki od gradova smije posjetiti samo jedanput. Postoje ceste koje spajaju bilo koja dva grada, a dužine u km dane su u tablici 2-1.

Tablica 2-1. Udaljenost između naselja na otoku Braču u km

	Supetar	Pučišća	Sumartin	Nerežišća
Supetar	0	9	29	39
Pučišća	9	0	24	32
Sumartin	29	24	0	30
Nerežišća	39	32	30	0

Zadatak je pronaći kružnu, najkraću rutu tako da se obiđu svi gradovi i vrati natrag u početni grad, a da ukupni prevaljeni put bude najkraći. Položaj gradova na karti otoka Brača prikazuje slika 2-3.

[22] U poglavlju 4.4.1 Produkcijski sustavi nalazi se Prolog kod za rješavanje ovog problema.



Slika 2-3. Karta otoka Brača s položajem naselja

Postavljanje zadatka

Ovo je jedan od klasičnih zadataka nazvan *problem trgovačkog putnika* za koji postoje i različita rješenja temeljena na teoriji grafova. U duhu umjetne inteligencije zadatak postavljamo na način da definiramo:

→***prostor stanja*** rješavanja zadatka su gradovi koje se treba obići i koje ćemo simbolički označiti slovima S – Supetar, B – Nerežišća, D – Pučišća, G – Sumartin (ove simboličke oznake koristimo sa pojednostavljene cestovne mreže otoka Brača prikazane na slici 3-8, na kojoj u sljedećem poglavlju detaljno objašnjavamo postupke pretraživanja). Gradovi dolaze u čvorove stabla problema, a veze između čvorova predstavljaju udaljenosti od gradova u km.

→***početno stanje*** - trgovac kreće iz grada S – Supetra

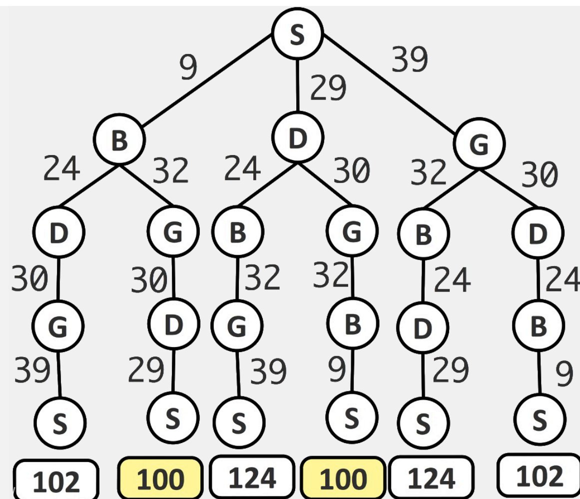
→***ciljno stanje*** - trgovac se treba vratiti u grad S i proći sve ostale gradove samo jedan put

→***skup pravila*** - ovdje nema nekih posebnih pravila. Jedino pravilo po kojem gradimo prostor rješenja zadatka je da sljedeći grad na nižoj razini ne smije biti niti jedan grad koji je u toj liniji trgovac već prošao s obzirom na to da smo u postavljanju zadataka definirali da trgovac ne smije kroz isti grad proći dva puta.

Rješavanje zadatka

Najjednostavniji sustavni pristup rješavanju ovog zadatka je postavljanje prostora stanja rješavanja zadatka u obliku stabla koje sadrži sve moguće gradove i to u slijedu kako ih trgovac treba posjetiti. Uz to za svaku rutu zbrajamo prijeđene udaljenosti, te nađemo najkraću. Izgled stabla rješenja prikazuje slika 2-4.

Zadatak ima dva rješenja: put Supetar (S) – Nerežišća (B) – Sumartin (G) – Pučišća (D) – Supetar (S) i put Supetar (S) – Pučišća (D) – Sumartin (G) – Nerežišća (B) – Supetar (S) s obzirom da je dužina u oba slučaja 100 km. Ova sustavna metoda izgradnje cijelog prostora rješenja je uspješna samo dok je broj gradova mali. Ako postoji N gradova, potrebno je istražiti (N-1)! različitih putova. To je za 4 grada $1{\cdot}2{\cdot}3 = 6$ putova, dok se za 11 gradova broj kombinacija penje na 3.628.800, a za 50-ak gradova broj kombinacija je blizu 10^{64}. Možda ovaj broj na prvi pogled i ne izgleda velik, ali ako spomenemo da je prema mišljenju kozmologa svemir star oko 10^{16} sekunda onda 10^{64} poprima sasvim drugačiju dimenziju. Ovakav nagli rast kombinacija naziva se ***kombinacijska eksplozija***. Ona predstavlja ograničenje sustavnom rješavanju zadatka. Još veći problem kombinacijske eksplozije je igra šaha.



Slika 2-4. Prostor rješenja za zadatak trgovačkog putnika koji treba proći kroz pet gradova

U prvom koraku imamo oko 15-ak mogućih poteza, u drugom približno 15^2, u trećem 15^3 čime broj vrtoglavo raste. Za rješavanje ovakvog tipa zadatka potrebno je definirati potpuno drugačiju strategiju upravljanja koja će dovesti do cilja bez potrebe da se prođu sve moguće kombinacije. Predložena će strategija do cilja dovesti jako brzo, a jedini je problem što nije sigurno da će dati najbolje rješenje, odnosno najkraći put. Dat će rješenje koje se koji put poklopi s najboljim, a koji put ne, ali će i tada biti relativno blizu najboljeg rješenja. Postupak se naziva ***heurističko pretraživanje*** (engl. *Heuristic Search*).

Sama riječ ***heuristika*** grčkog je porijekla i dolazi od riječi *„heurisko"* (εὑρίσκω) što znači *otkrio sam*. To je također i korijen poznate riječi *„Eureka!"*, Arhimedovog uzvika *„Pronašao sam!"* kada je otkrio metodu određivanja čistoće zlata. Heurističko pretraživanje je postupak kojim se povećava efikasnost postupka traženja, obično uz žrtvovanje potpunosti. Postoji niz općenitih heuristika koje su korisne za rješavanje velikog broja zadataka, dok s druge strane postoje i specijalizirane heuristike razvijene za potrebe rješavanja zadataka određene, uske domene.

Primjer univerzalne heuristike koristan pri rješavanju kombinacijskih problema, kao što je naš zadatak trgovačkog putnika, je ***algoritam najbližeg susjeda*** (engl. *Nearest Neighbour Algorithm*). U našem slučaju on se sastoji samo od dva koraka:

1. Otiđi u najbliži grad.

2. Ponovi prvo pravilo dok se ne obiđeš sve gradove.

Prema slici 2-4 to bi značilo iz Supetra (S) otići u Nerežišća (B) s obzirom na to da su najbliža, zatim u Pučišća (D) i onda u Sumartin (G) koji je još ostao, pa natrag u Supetar (S). U ovom primjeru vidimo da algoritam najbližeg susjeda ne daje ujedno i jedno od najboljih rješenja (ukupno najkraći pređeni put). U nekoj drugoj situaciji ukupno prijeđeni put može biti i najkraći, ali heurističko pravilo to ne garantira.

Bez heuristike su zadaci u kojima se javlja kombinacijska eksplozija puno teže rješivi, ali postoje i drugi razlozi koji govore u prilog heuristike. Iako heuristika ne garantira najbolje rješenje, mi ga u biti i rijetko kada baš i trebamo. U svom svakodnevnom životu najčešće se rukovodimo ***prvim zadovoljavajućim rješenjem***. Dobar je primjer parkiranje automobila. Većina će ljudi prekinuti traženje parking mjesta koje će biti bliže njihovoj konačnoj destinaciji već kada nađu prvo slobodno mjesto koje je po njihovom sudu dovoljno blizu i zadovoljavajuće. Napomenimo da se o heuristici i heurističkom pristupu rješavanju zadataka pisalo još 1940-ih godina. Matematičar ***George Pólya*** je 1945. godine objavio knjigu ***„Kako riješiti"*** (engl. *How to solve it*)[23] u kojoj daje niz heurističkih naputaka kako riješiti

[23] https://math.hawaii.edu/home/pdf/putnam/PolyaHowToSolveIt.pdf

matematičke probleme, ali se njegove heuristike mogu upotrijebiti i kod šire klase problema. Ova je knjiga ostavila veliki trag na istraživače umjetne inteligencije. U sljedećem poglavlju navode se neki od Pólyaovih principa analize zadatka. Spomenimo samo da se prema Pólyau svako rješavanje zadatka treba provoditi u četiri koraka:

- shvatiti (razumjeti) zadatak
- napraviti plan rješavanja
- provesti plan rješavanja i na kraju
- provjeriti rezultat.

Shvatiti zadatak znači prije svega razumjeti što se traži, što je zadano i koja su ograničenja, ali ako se radi o kompleksnom zadatku, trebamo isto otkriti osnovne značajke ili karakteristike zadatka.

2.2 Osnovne značajke zadataka

Već smo naglasili da je osnovni postupak rješavanja zadataka unutar umjetne inteligencije postupak ***pretraživanja***. Pretraživanje je skup univerzalnih metoda primjenjiv pri rješavanju široke klase problema, a uključuje primjerice već spomenuto heurističko pretraživanje, slijepo i usmjereno pretraživanje o kojima ćemo u 3. poglavlju detaljnije govoriti. Međutim, svaka od tehnika pretraživanja uključuje i niz specifičnih tehnika primjenjivih za određeni tip zadataka. Kako bi se mogla izabrati odgovarajuća metoda ili kombinacija nekoliko različitih metoda, potrebno je analizirati zadatak uzimajući u obzir njegove osnovne značajke. Najvažnije od njih su:

1. ***Dekompozicija*** – može li se zadatak razbiti u više nezavisnih podzadataka lakših za rješavanje?
2. ***Zanemarivanje*** – mogu li se zanemariti neki od koraka rješavanja zadataka?
3. ***Predvidivost*** – je li prostor rješenja zadatka predvidiv?
4. ***Očitost rješenja*** – je li dobro rješenje očito u usporedbi s ostalim mogućim rješenjima?
5. ***Dosljednost znanja*** – je li baza znanja koja je potrebna za rješavanje zadatka dosljedna (konzistentna), bez kontradikcija?
6. ***Uloga znanja*** – koja je uloga znanja u rješavanju zadatka? Je li znanje neophodno potrebno za rješavanje zadatka ili je potrebno samo za olakšavanje rješavanja zadatka?
7. ***Interaktivnost*** – zahtijeva li zadatak interakciju čovjeka i računala tijekom rješavanja?

Pogledajmo ukratko svaku od ovih značajki.

2.2.1 Dekompozicija

Ako je moguće zadatak razbiti u niz jednostavnijih zadataka, tada se i jednostavnije metode mogu primijeniti za njegovo rješavanje. Ovo je standardni postupak koji se primjenjuje kod rješavanja matematičkih zadataka. Tipičan primjer je rješavanje integrala koji obično rastavljamo na osnovne integrale koji su tabličnog oblika. Kod rješavanja integrala to možemo raditi zato što vrijedi zakon superpozicije, ali su kompleksni zadaci često takvi da suma parcijalnih rješenja ne odgovara ukupnom rješenju pa je dekompoziciju često nemoguće napraviti.

2.2.2 Zanemarivanje

Važna karakteristika zadatka je može li se pojedini korak u rješavanju ***zanemariti ili se treba ponoviti***. Zadaci kod kojih se pojedini korak može zanemariti mogu se uspješno rješavati upravljačkom strukturom koja se nikada ne vraća unatrag, pa je stoga jednostavna i brza. Zadaci koji zahtijevaju ponavljanje svih koraka imat će složeniju strukturu upravljanja, koja će dopuštati vraćanje unatrag ako se napravi pogrešan potez, dok se kod neponovljivih problema pogrešan potez ne smije napraviti, zato što

nema popravljanja. Na primjer, ako u zadatku dva vrča naiđemo na neko stanje koje smo već imali, možemo ga zanemariti i time skratiti proces traženja rješenja.

2.2.3 Predvidljivost

Uz značajku zanemarivanja važna je i značajka ***predvidljivosti*** prostora rješenja zadatka. Pitanje je jesu li buduća rješenja predvidiva s određenim stupnjem sigurnosti ili su nepredvidiva. Planiranjem postupka rješavanja prva klasa problema može se relativno uspješno riješiti, dok se kod druge klase u najboljem slučaju može garantirati s većom ili manjom sigurnošću da će se zadatak moći riješiti. Najteži zadaci su ***neponovljivi*** **i** ***nepredvidivi***.

Tipičan primjer takvih zadataka su različite igre u kojima sudjeluju najmanje dva igrača pa sljedeće stanje ovisi o reakciji protivničkih igrača. U ovu grupu mogu spadati pravni savjetodavni (ekspertni) sustavi koji bi npr. trebali predložiti postupak obrane, ili pak zadaci upravljanja samohodnim robotom u nepoznatom ambijentu.

2.2.4 Očitost rješenja

Sljedeće je pitanje je li rješenje ***apsolutno dobro*** ili ***relativno dobro***. U odnosu na nj razlikujemo tzv. zadatke ***bilo kojeg puta*** (engl. *Any-Path Problem*) i zadatke ***najboljeg puta*** (engl. *Best-Path-Problems*). Prvi su lakši za rješavanje zato što je svako rješenje, bez obzira na put do njega, dobro rješenje i ne treba se uspoređivati s rješenjem koje je dobiveno na neki drugi način. Zadatak bilo kojeg puta je primjer dva vrča u slučaju da dolazimo do rješenja s istim brojem koraka. Oba rješenja koja smo naveli su jednako vrijedna i potpuno nam je svejedno do kojeg smo prvo došli. Zadatak trgovačkog putnika je zadatak tipa najboljeg puta. Ne možemo garantirati da smo došli do najboljeg rješenja dok ne prođemo sve kombinacije. To što zadatak ima dva najkraća puta za postupak rješenja nije bitno. Da smo stvarno prešli najkraći put, znamo tek nakon što testiramo sva moguća rješenja.

2.2.5 Dosljednost znanja

Dosljednost (konzistentnost) baze znanja također je važna značajka koja utječe na postupak rješavanja. Na primjer, ako baza znanja sadrži logičke tvrdnje da je A točno i da A nije točno, tijekom rješavanja zadatka i izgradnje prostora rješenja u nekom trenutku nećemo znati kako postupiti. Znanje potrebno za prijelaz iz jednog stanja u drugo nije dosljedno.

Posebno je kritična situacija u vezi dosljednosti znanja ako znanje prikazujemo klasičnom, na primjer predikatnom logikom. Predikatni račun nije prilagođen nedosljednoj bazi znanja. Kritična je i situacija u kojoj imamo sustav koji uči kod kojega postoji mogućnost dodavanja novoga znanja u bazu znanja. Novo, dodano znanje ne smije biti u kontradikciji s nekim već postojećim znanjem. Ako je u kontradikciji, onda jednu od kontradiktornih tvrdnji trebamo izbrisati. U okviru istraživanja vezanih s provjerom dosljednosti baze znanja razvijen je posebni ***sustav održavanja istinitosti*** (engl. *Truth Maintenance System*) čiji je zadatak provjeravati dosljednost baze znanja nakon dodavanja novog znanja.

2.2.6 Uloga znanja

Ponekad je baza znanja ***nužna za rješavanje zadatka***. Bez nje se ne može razvijati prostor rješenja niti se može iz jednog stanja prelaziti u drugo. Tipičan je primjer zadatak dva vrča. Ako ne znamo što s vrčevima možemo napraviti, ne možemo niti riješiti zadatak. Pretpostavimo li da zadatak zadamo nekome tko nikada u životu nije vidio vrč ili posudu i ne zna da se vrč može puniti, on ne može niti riješiti zadatak. Pravila o punjenju i pražnjenju vrčeva bitan su dio postupka rješavanja.

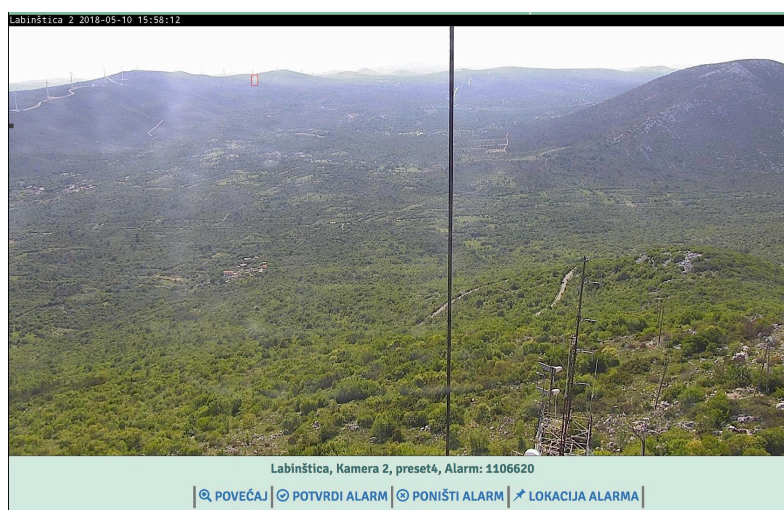
S druge strane postoje zadaci kod kojih znanje služi samo za ***ubrzavanje postupka rješavanja***, npr. ako bismo imali računalo neograničene brzine i kapaciteta, tada bi se zadatak trgovačkog putnika mogao riješiti bez posebnog znanja jednostavnim generiranjem svih mogućih putova. U ovom tipu zadatka znanje, na primjer u obliku heurističkog pravila, služi samo za ubrzavanje postupka rješavanja. Slična je situacija i kod igre šaha. Računalo neograničene brzine i kapaciteta bit će uspješno u igri šaha

samo na temelju osnovnog znanja o dozvoljenim potezima, međutim isto to računalo nikada ne bi moglo predvidjeti rezultate na izborima ako ima samo znanje o tome da se može glasati ili za stranku A ili za stranku B. Tu osnovno znanje nije dovoljno. Da bi rezultati predviđanja ishoda izbora bili suvisli, treba velika količina specijalističkog znanja.

2.2.7 Interaktivnost – uloga čovjeka

Posljednja karakteristika vezana je s načinom rješavanja i odnosi se na pitanje je li se zadatak rješava potpuno ***samostalno***, bez ikakve asistencije čovjeka, ili je zadatak takav da ga se rješava ***nesamostalno***, pa je u određenom koraku rješavanja zadatka bitan čovjek kao dodatni izvor informacija bez kojih se ne može doći do dobrog rješenja. Naravno da je prvi tip zadataka puno teži za rješavanje. Tipičan primjer samostalnog rješavanja zadatka je kretanje samohodnog robota po nepoznatom terenu bez asistencije čovjeka, a tipičan primjer nesamostalnog rada su razni savjetodavni dijagnostički sustavi ili detekcijski sustavi. Na *Fakultetu elektrotehnike, strojarstva i brodogradnje Sveučilišta u Splitu* razvijen je sustav za rano otkrivanje šumskog požara analizom slike u vidljivom dijelu spektra ***OIV Fire Detect AI*** [24] temeljen na različitim postupcima umjetne inteligencije.

Da je sustav potpuno samostalan, on bi nakon detekcije automatski podigao požarnu uzbunu i pokrenuo postupak gašenja. Međutim, kako ovakvi sustavi uvijek imaju određeni broj lažnih alarma, odmah je na početku osmišljen kao nesamostalni, poluautomatski sustav. Ako ovaj sustav detektira mogući šumski požar, prezentira ga čovjeku operateru koji donese konačnu odluku o tome radi li se o požaru ili ne. Da bi ostvario svoj cilj, a to je rana detekcija šumskog požara, nužno mu je dodatno znanje čovjeka koji kaže: *„Da, to je stvarno požar"* i klikne na naredbu POTVRDI ALARM ili kaže: *„Ne, to nije požar"* i klikne na naredbu PONIŠTI ALARM (slika 2-5).



Slika 2-5. Operatorski ekran inteligentnog sustava za rano otkrivanje šumskog požara. Za ispravno funkcioniranje sustava nužna je interaktivnost s operatorom koji donosi konačnu odluku o tome je li na slici požar ili nije klikom na odgovarajuću naredbu.

2.3 Rješavanje zadatka metodama umjetne inteligencije

Već smo naglasili da su osnovne metode umjetne inteligencije za rješavanja zadataka tzv. ***slabe metode*** koje se ne zasnivaju na formuli ili strogom matematičkom algoritmu. Osnovni postupak rješavanja naziva se ***pretraživanje***, a pretražuje se prostor stanja u kojem se nalaze moguća rješenja zadatka kojem se traži put od početnog stanja do ciljnog (ili ciljnih) stanja. U prethodnim smo poglavljima upoznali dva tipična tipa zadataka umjetne inteligencije čijem se rješavanju može pristupiti postupcima

[24] Za detalje pogledajte https://oiv.hr/en/services-and-platforms/oiv-fire-detect-ai

umjetne inteligencije. U problemu dva vrča prostor rješenja zadataka gradili smo postupno iz koraka u korak, a u svakom koraku iz baze znanja (osam pravila punjenja/pražnjenja) tražili smo pravila koja možemo primijeniti u tom trenutku. Kako ćemo razvijati prostor rješenja ili, kako je uobičajeno u terminologiji umjetne inteligencije reći, kako ćemo ***razvijati čvor*** (engl. *Expanding the Node*) stabla rješenja zadatka, ovisi o ***strategiji pretraživanja*** što je tema sljedećeg poglavlja. I u drugom zadatku koji je predstavljao problem trgovačkog putnika gradili smo prostor rješenja zadatka samo što ga je sada činio slijed posjećivanja gradova s liste. Nisu postojala pravila prelaska iz jednoga stanja u drugo zato što se iz bilo kojeg grada moglo voziti u bilo koji drugi grad. Strategija pretraživanja u ovom je slučaju definirala u koji ćemo sljedeći grad od svih preostalih neposjećenih gradova otići. Problemu trgovačkog putnika sličan je ***problem najkraćeg puta*** (engl. *Shortest Path Problem*) kada se traži najkraći put između dva grada i ***problem kineskog poštara*** (engl. *Chinese Postman Problem*) kada se traži zatvoreni put kojim ćemo proći sve ceste koje spajaju gradove s tim da kroz svaki grad prođemo barem jedanput. Svi su ovi zadaci pogodni za rješavanje primjenom odgovarajuće strategije pretraživanja, a različitim strategijama pretraživanja detaljno se bavi sljedeće poglavlje.

....



Dodatak: U poglavlju o interaktivnosti i ulozi čovjeka u donošenju konačne odluke spomenuli smo ***inteligentni sustav*** za rano otkrivanje šumskog požara analizom slike u vidljivom dijelu spektra ***OIV Fire Detect AI***, koji je razvijen na *Fakultetu elektrotehnike, strojarstva i brodogradnje Sveučilišta u Splitu*. Kako se sustav temelji na brojnim inteligentnim tehnologijama, ovdje ćemo malo više o njemu kazati. Kako se radi o ***sustavu,*** to znači da ima više sastavnih dijelova koje možemo izdvojiti iz okružja u kojem djeluje i da ima svoju svrhu. Svaki inteligentni sustav mora imati tri osnovne komponente. Prvi dio su ***osjetila*** (senzore) kojima dobiva sliku o okružju u kojem djeluje. Drugi dio je ***centralni dio*** u kojem se ulazne senzorske informacije obrađuju, analiziraju, te se na temelju njih i pohranjenog znanja donose odluke o djelovanju. Ova se informacija prenosi trećem dijelu kojeg čine ***izvršne strave*** (aktuatori) pomoću kojih inteligentni sustav povratno djeluje na svoje okružje. Inteligentni sustav može biti ***virtualni*** i ***realni***. Virtualni inteligentni sustav djeluje u virtualnom okružju (na primjer Internet), pa su mu i osjetila i izvršne sprave virtualni, a ulazne i izlazne informacije su datoteke. Realni inteligentni sustav djeluje u stvarnom, realnom okružju u kojem i mi djelujemo, pa su mu osjetila i izvršne naprave tehnički uređaji. Osnovna svrha ***OIV Fire Detect AI*** je upozoravanje na pojavu požara otvorenog prostora u nastajanju i predviđanje njegovog mogućeg širenja, pa su mu osjetila visokorezolucijske, panoramske kamere i meteorološke stanice, a izvršne sprave naprave za podizanje alarma (vizualnog i akustičkog). U centralnom dijelu sustava analiziraju se ulazne slike koristeći različite napredne postupke digitalne obrade i analize slike, te se na temelju njih u dijelu za zaključivanje, na temelju različitih mehanizama zaključivanja, donosi odluka da li su na slici značajke požara otvorenog prostora (pojava dima i/ili vatra) ili nisu. Ukoliko je zaključak pozitivan podiže se alarm i upozorava osoblje o mogućnosti pojave požara. Kako sustav radi u kooperaciji s čovjekom operaterom, na njemu je, nakon provjere, donijeti konačnu odluku da li se stvarno radi o požaru ili ne. Ako se požar potvrdi operater može pokrenuti i simulator mogućeg kretanja požara slijedećih nekoliko sati, naravno ukoliko ne dođe do intervencije gašenja požara. Sustav za sada pokriva područje Dalmacije sa stotinjak kamera.

3 RJEŠAVANJE ZADATAKA METODAMA PRETRAŽIVANJA



”

Pretraživanje je jedan od temeljnih načina kako se rješavaju kompleksni, nealgoritamski zadaci postupcima umjetne inteligencije. Pretražuje se detaljno i sustavno prostor mogućih rješenja zadatka (prostor stanja) unutar kojeg nastojimo pronaći put od početnog stanja iz kojeg krećemo do ciljnog ili ciljnih stanja koje nastojimo doseći. Iako možda na prvi pogled postupci pretraživanja i ne izgledaju previše „pametni“, uspjeli su riješiti brojne kompleksne zadatke.

Metode rješavanja zadataka koje nazivamo ***pretraživanje*** (engl. Search) definirane su kao detaljno pretraživanje prostora stanja (prostora mogućih rješenja zadatka) gdje tražimo put od početnog stanja do ciljnog (ili ciljnih) stanja[25]. Pretraživanje je osjetljivo na kombinacijsku eksploziju, pa spada u ***slabe metode***. Iako su ove metode „slabe”, u posljednjih su nekoliko desetljeća riješile brojne „jake” probleme, pa su još uvijek temeljne metode rješavanja zadataka u brojnim sustavima umjetne inteligencije, od problema traženja najboljeg puta, do računalnih realizacija igranja različitih igara, primjerice šaha.

Prostor rješenja zadataka pretraživanjem može se predstaviti usmjerenim grafom tipa stabla, kod kojega krećemo od početnog stanja te prolazimo kroz niz međustanja nastojeći doći do jednog od ciljnih stanja. Metode pretraživanja na sustavni način konstruiraju graf rješenja zadatka. Pri tome se graf može konstruirati ***eksplicitno***, na početku rješavanja zadatka, pa se postupak traženja puta od početnog do ciljnog stanja provodi po cijelom prostoru stanja, ili ***implicitno***, dio po dio kako se zadatak rješava i to samo u dijelu važnom za traženje rješenja.

U svakoj od metoda pretraživanja važne su sljedeće osobine:

- ***način predstavljanja prostora stanja*** (prostora problema, prostora rješenja) i to na formalni, matematički način koji se može jednostavno prenijeti računalnom programu
- ***izbor prikladnog pravila*** za prelazak iz stanja u stanje koje se dohvaća iz baze znanja u kojoj su pohranjena sva dostupna pravila, zapisana na formalni, matematički način i

[25] Engleska riječ *„search”* prevodi se na hrvatski i kao *pretraživanje* i kao *traženje*, ali ove riječi u hrvatskom jeziku imaju različito značenje. Glagol *„tražiti”* prema Hrvatskom jezičnom portalu (http://hjp.znanje.hr) definira se (među ostalim) kao *„truditi se da se pronađe ono što je izgubljeno, zametnuto ili još nepoznato”*, dok glagol *„pretražiti”* znači *„detaljno pregledati, tražeći što”*. Kako se u kontekstu rješavanja zadataka ovom metodom radi *„o sustavnom traženju rješenja u prostoru stanja*”, engleskoj riječi *„search”* više bi odgovarala hrvatska riječ ***„pretraživanje”***.

- ***vođenje postupka pretraživanja*** na sustavan način koji omogućava napredak od početnog prema ciljnim stanjima.

Metode pretraživanja primjenu nalaze kod rješavanja brojnih problema koje grubo dijelimo u dvije grupe:

- ***ilustrativni problemi*** ili kako se uobičajeno nazivaju ***problemi za igru*** (engl. *Toy Problems*) koji nemaju neku posebnu praktičnu primjenu, već prije svega služe za ilustraciju i objašnjenje različitih postupaka, te
- ***stvarni problemi*** ili kako se uobičajeno zovu ***problemi stvarnog svijeta*** (engl. *Real-World Problems*) koji nalaze stvarnu primjenu u različitim ljudskim djelatnostima.

Drugi način podjele je na:

- ***problemi jednog agenta-igrača*** (engl. *Single-agent Problem*)
- ***problemi dva agenta-igrača*** (engl. *Two-players Game*) i
- **problemi zadovoljenja ograničenja** (engl. *Constraint-satisfaction Problems*).

Kod ***problema jednog agenta*** slijed događanja ovisi samo o agentu (igraču) i njegovim potezima. Zadatak mu je od nekog početnog stanja doći do konačnog stanja na bilo koji način ili na najbolji mogući način. Tipičan primjer je ***problem traženja puta*** (engl. *Road Routing Problem*) kod kojeg zadatak može biti samo pronaći put od početne točke do krajnje točke, ali i pronaći najkraći put od početne do krajnje točke.

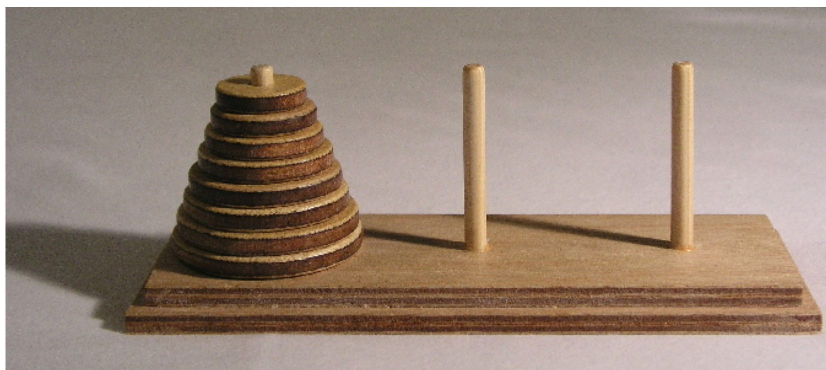
Problemi dva agenta su različite igre, na primjer kružić-križić, šah, go. Kod sviju njih sljedeći potez prvog agenta (igrača) ovisi o potezu drugog agenta (igrača), pa je kompleksnost ovakvih zadataka puno veća.

Problemi zadovoljenja ograničenja svode se na to da skup objekata treba zadovoljiti određeni broj ograničenja. Tipični primjeri takvih problema su problemi osam kraljica ili problemi bojanja grafa koje ćemo u nastavku posebno opisati.

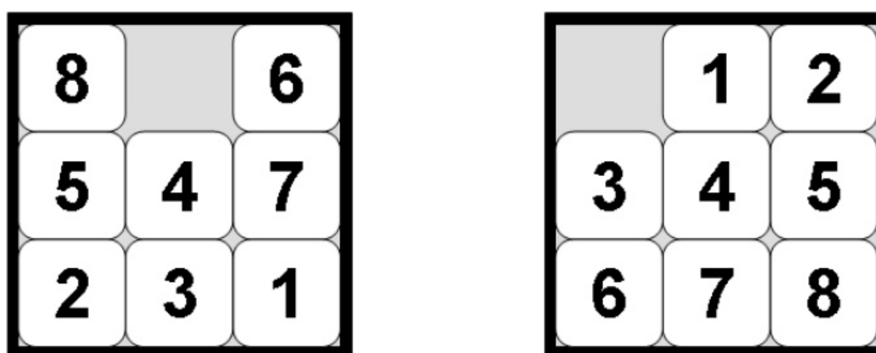
Ilustrativni problemi obično su dobro zadani i precizno opisani, pa se na njima mogu testirati uspješnosti različitih postupaka pretraživanja, te se zbog toga uobičajeno koriste u udžbenicima iz kojih se uče temelji umjetne inteligencije. Stvarni su problemi dosta složeniji, ali se u biti temelje na istim algoritmima, samo s većim stupnjem složenosti.

Primjeri ***ilustrativnih problema jednog agenta*** su:

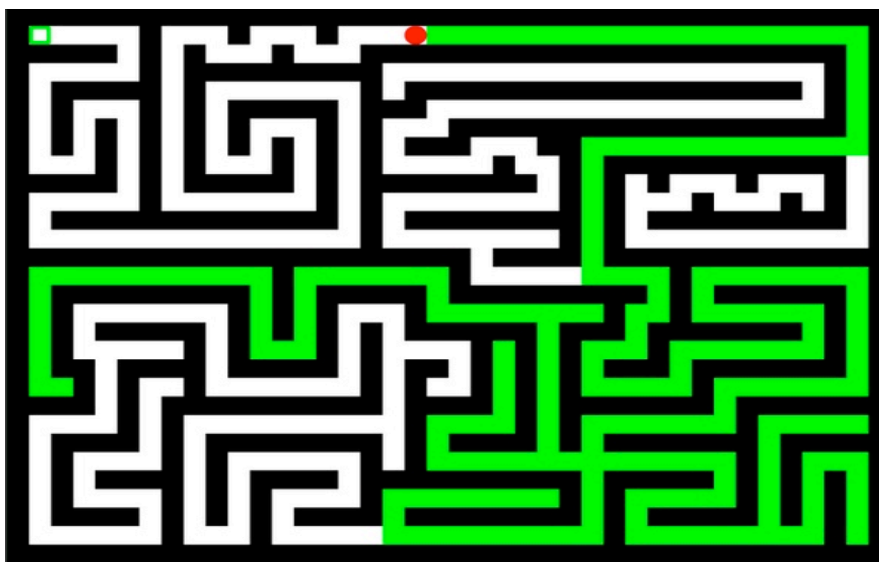
- Različite *„**mozgalice**"* – problemski zadaci kod kojih se treba riješiti određeni problem zadan pričom – tipičan je ***problem dva vrča*** koji smo već upoznali u prethodnom poglavlju, u kojem je zadatak iz nekog početnog stanja napunjenosti vrčeva doći do konačnog stanja koristeći dozvoljena pravila prelijevanja. Ovaj se problem može i zakomplicirati na *n* vrčeva. Drugi primjer je ***problem farmera, lisice, guske i zrnja*** (engl. *Farmer, Fox, Goose and Grain Problem*) kod kojeg farmer treba prevesti preko vode lisicu, gusku i zrnje, ali pri tome u čamcu ne smiju biti lisica i guska ili guska i zrnje. Treći primjer su ***tornjevi Hanoia*** (engl. *Tower of Hanoi*) kod kojeg se diskovi različitog promjera poredani na jednom od stupova trebaju prebaciti na drugi, na način da nikada veći disk ne ide iznad manjega, a kod prebacivanja se može koristiti pomoćni stup (slika 3-1), itd.
- ***Problem slagalice s praznim mjestom*** (engl. *Sliding Puzzle* ili *Sliding Block Puzzle*) koja može imati *n x n* polja kod koje iz nekog početnog stanja pomičući polja horizontalno ili vertikalno trebamo doći do ciljnog stanja (primjer slagalice 3 x 3 je na slici 3-2).
- ***Problem labirinta*** (engl. *Maze Problem*) kod kojeg trebamo pronaći put od ulaza do izlaza iz labirinta. Labirint spada u ilustrativne, školske probleme, ali danas nalazi i direktnu primjenu u mobilnoj robotici.



Slika 3-1. Problem tornjeva Hanoia[26]



Slika 3-2. Problem slagalice s praznim mjestima veličine 3 x 3 s primjerom početnog stanja i željenog (ciljnog) stanja

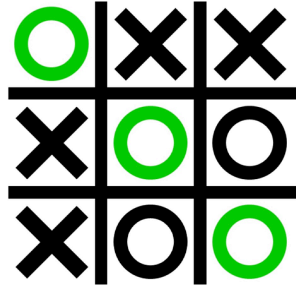


Slika 3-3. Problem labirinta s primjerom sustavnog pretraživanja s ciljem pronalaska puta od ulaza do izlaza labirinta

[26] Slika s https://commons.wikimedia.org/wiki/File:Tower_of_Hanoi.jpeg uz CC BY-SA 3.0 licencu.

Primjeri ***ilustrativnih problema dva agenta*** su različite igre dvaju igrača:

- ***Kružić-križić*** (engl. *Tic-Tack Toe*) – igra dvaju igrača koju smo sigurno svi barem jednom u životu igrali. Cilj je postaviti tri svoja znaka horizontalno, vertikalno ili dijagonalno.



Slika 3-4. U igri kružić-križić jedan igrač nastoji postaviti tri svoja znaka u horizontali, vertikali ili dijagonali, pa je u primjeru sa slike pobijedio kružić

- ***Šah***, možda jedan od najpoznatijih primjera testiranja algoritama umjetne inteligencije, posebno nakon što je 1997. godine računalni program za igranje šaha poznat pod nazivom ***IBM's Deep Blue*** pobijedio ***Garija Kasparova***, višestrukog svjetskog prvaka 3½ – 2½. Iako je za neke ova pobjeda bila diskutabilna, činjenica je da je 2006. godine drugi računalni program za igranje šaha ***Deep Fritz*** pobijedio tadašnjeg svjetskog prvaka ***Vladimira Kramnika*** 4 : 2.
- ***Go***, apstraktna, 2500 godina stara strategijska igra dvaju igrača, igra jednostavnih pravila, ali puno kompleksnija od igre šaha, pa je tek 2015. godine računalni program ***AlphaGo*** pobijedio profesionalnog igrača *Go*-a bez hendikepa na punoj 19 x 19 ploči. 2016. godine ***AlphaGo*** pobjeđuje korejskog igrača ***Leeja Sedola***, u to vrijeme najboljeg svjetskog igrača *Go*-a rezultatom 4 : 1. Više u dodatku iz ovog poglavlja.

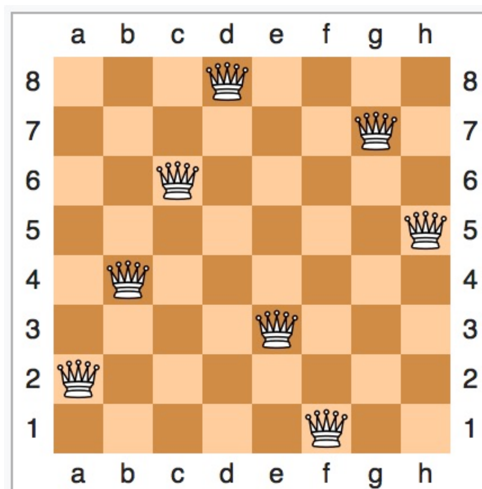


Slika 3-5. Go – drevna apstraktna strategijska igra dvaju igrača porijeklom iz Kine – puno kompleksnija od igre šaha, pa je tek 2015. godine računalni program AlphaGo pobijedio profesionalnog igrača Go-a na punoj ploči bez hendikepa[27]

[27] Slika je s web-stranice https://commons.wikimedia.org/wiki/File:13_by_13_game_finished.jpg uz CC BY-SA 2.0 licencu.

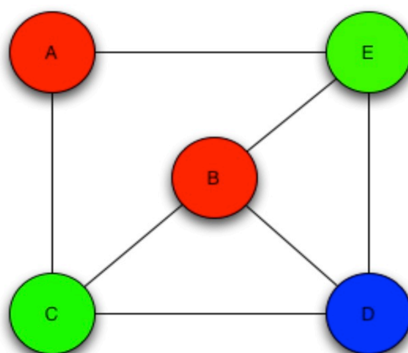
Ilustrativni problemi koji spadaju ***u probleme zadovoljenja ograničenja*** su:

- ***Problem osam kraljica*** (engl. *Eight Queens Puzzle*) kod kojeg treba na šahovsku ploču 8 x 8 postaviti 8 kraljica na način da se kraljice međusobno ne ugrožavaju. ***Edsger Dijkstra*** je 1972. godine baš ovaj problem koristio kod opisa strukturnog programiranja uz detaljni opis tzv. *backtracking* algoritma temeljenog na dubinskom pretraživanju.



Slika 3-6. Problem osam kraljica – kraljice treba postaviti na šahovsku ploču tako da se međusobno ne ugrožavaju

- ***Problem bojanja grafa*** (engl. *Graph Coloring Problem*) (2D ili 3D) ima dvije inačice: bojanje vrhova grafa na način da dva vrha povezana stranicom nemaju istu boju ili bojanje stranica grafa kada dvije stranice povezane istim vrhom ne smiju biti bojane istom bojom.

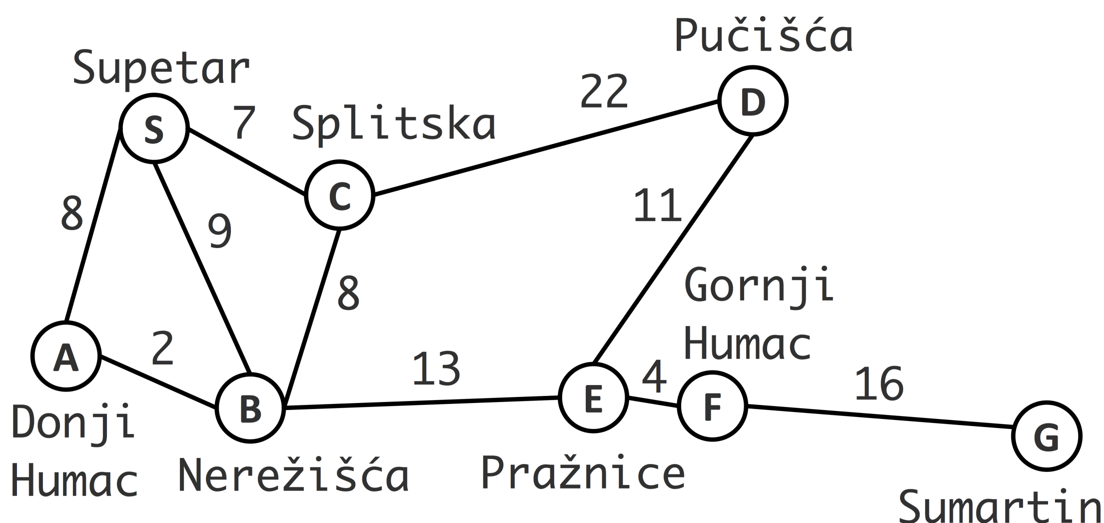


Slika 3-7. Problem bojanja grafa

- Različite ***„slagalice"***, na primjer križaljke, sudoku, kakuro, hidato, ali i logičke zagonetke. Posebno zanimljiv primjer logičkih zagonetki su ***zagonetke logičke tablice*** (engl. *Logic Grid Puzzles*), na primjer ***Einsteinova zagonetka*** (engl. *Einstein's Riddle*). Temelji se na tome da se postavi određeni broj logičkih izjava (tvrdnji) vezanih uz stanare 5 kuća, njihovih omiljenih pića, cigareta i ljubimaca. Na primjer *„Postoji pet kuća."*, *„Englez živi u crvenoj kući."*, *„Španjolac ima psa."* itd. Jedan od kućnih ljubimaca je ribica, pa je pitanje na koje se treba odgovoriti: *"Tko ima ribicu za ljubimca?"* Druga varijanta ove zagonetke je malo drugačija i naziva se ***zebra zagonetka*** (engl. *Zebra Puzzle*) zato što se traži tko ima zebru za ljubimca. U dodatku ovog udžbenika, u dijelu koji se bavi programskim jezikom Prolog, zagonetka je detaljno objašnjena i riješena u Prologu.

Kod ***problema stvarnog svijeta*** koriste se isti algoritmi, ali se primjenjuju za rješavanje stvarnih problema u pojedinim područjima. Tipični primjeri su:

- ***Problem pronalaska puta*** (engl. *Route Finding Problem*) od neke početne pozicije do konačnog odredišta. Ovaj smo tip problema već spominjali u obliku ***problema najkraćeg puta, problema trgovačkog putnika*** *i* ***problema kineskog poštara***. ***Problem najkraćeg puta*** (engl. *Shortest Path Problem*) često se koristi kao primjer na kojem se ilustriraju razlike različitih algoritama pretraživanja, pa ćemo ga i u ovom udžbeniku koristiti na primjeru otoka Brača. Slika 2-3 prikazuje cestovnu mrežu otoka Brača koju smo koristili kod objašnjavanja ***problema trgovačkog putnika***, dok ovdje ilustriramo ***problem najkraćeg puta***, ali na pojednostavljenoj cestovnoj mreži otoka Brača na kojoj smo istakli 8 naselja: Supetar (S), Donji Humac (A), Nerežišća (B), Splitska (C), Pučišća (D), Pražnice E), Gornji Humac (F) i Sumartin (G). Slika 3-8 prikazuje zadatak i formalno u obliku otežanog grafa, gdje su čvorovi gradovi simbolički prikazani slovima, a težine na granama predstavljaju zaokružene udaljenosti između njih iskazane u kilometrima. Zadatak je pronaći najkraći put između dvaju gradova – u našem slučaju to će biti najkraći put između Supetra i Sumartina. Postoji ukupno 6 različitih putova, a samo je jedan od njih najkraći. Ovaj ćemo primjer koristiti kod opisa različitih algoritama pretraživanja.



Slika 3-8. Cestovna mreža otoka Brača i formalno prikazani problem najkraćeg puta na pojednostavljenoj mreži s 8 naselja otoka Brača

- ***Upravljanje digitalnim prometom*** (engl. *Routing Problem*) od izvora do konačnog odredišta preko mrežnih usmjerivača, komutatora i koncentratora.
- ***Kod projektiranja integriranih sklopova, posebno VLSI*** (engl. *Very-Large-Scale Integration*), algoritmi pretraživanja se koriste kod pozicioniranja milijuna komponenti i priključaka na čipu kako bi se smanjila površina i skratilo kašnjenje signala.
- ***Robotska navigacija*** koja je zapravo prebacivanje problema labirinta u realni prostor. Danas, sa sve većom pojavom autonomnih vozila, ovo područje postaje sve zanimljivije.
- ***Automatsko sekvenciranje montaže*** (engl. *Automatic Assembly Sequencing*) čiji je cilj pronaći redoslijed za sastavljanje složenog objekta od sastavnih dijelova koji ponekad mogu biti i geometrijski složeni.
- ***Dizajn proteina*** kojem je zadatak utvrđivanje niza aminokiselina koje će se presaviti u 3D protein s točno određenim svojstvima za liječenje neke bolesti.

- ***Pronalaženje grešaka u kodu*** (engl. *Code Debugging*) – ovo je područje u posljednjih desetak godina posebno zanimljivo, a cilj mu je automatska analiza programskog koda i pronalaženje mogućih pogrešaka.

Primjera je je još puno, ali kada se shvati kako algoritmi pretraživanja funkcioniraju, lako ih je i primijeniti. U nastavku spominjemo neke zajedničke karakteristike postupaka pretraživanja, a nakon toga prikazujemo osnovne tipove algoritama pretraživanja, od jednostavnih do složenih.

3.1 Predstavljanje prostora problema (prostora pretraživanja)

Prostor problema, ili kako se češće zove ***prostor pretraživanja*** (engl. *Search Space*), je prostor u kojem tražimo rješenje problema. Ovaj prostor grafički predstavljamo grafom tipa stablo koji nazivamo ***stablo rješenja problema***. U slučaju problema punjenja dvaju vrčeva dio stabla problema prikazuje slika 2-2, a kod problema trgovačkog putnika cjelovito stablo problema prikazuje slika 2-4. Stablo problema gradi se postupno i sustavno od početnog stanja do ciljnog kod unaprijednog pretraživanja i obrnuto, od ciljnog stanja do početnog kod povratnog pretraživanja. Postupci pretraživanja razlikuju se upravo po tome kako se gradi stablo rješenja problema, ali bez obzira o kojem se postupku radi, obavezno treba biti ugrađen mehanizam koji ne dozvoljava stvaranje petlji i vraćanje unatrag. Na primjeru problema dvaju vrčeva sa slike 2-2 to na primjer znači da smo na trećoj razini naišli na stanje (0,0) od kojeg smo krenuli, pa se to stanje dalje ne razvija. Nadalje, ako na istoj razini postoje dva identična stanja, na primjer stanja (4,3) na trećoj razini problema dvaju vrčeva, dalje se razvija samo jedno od njih (obično prvo).

Postupak sustavnog građenja stabla rješenja problema ima i svoju specifičnu terminologiju:

- ***čvor*** (engl. *Node*) – moguće rješenje u problemskom prostoru pretraživanja u kojem odlučujemo kojim putem ići dalje
- ***poveznica*** (engl. *Link* ili *Edge*) – veza između dvaju čvorova kojoj može biti pridružena određena težina (npr. broj pravila koje smo primijenili u problemu dvaju vrčeva ili udaljenost između dvaju gradova u problemima pronalaženja puta)
- ***grana*** (engl. *Branch*) – put između više čvorova – sastoji se od čvorova kroz koje se prolazi i grana kojima se prolazi
- ***roditelji*** (engl. *Parents*) – čvorovi koji se u stablu rješenja problema nalaze iznad čvorova ***djece*** (engl. *Child*)
- ***predci*** (engl. *Anecestors*) – svi roditeljski čvorovi na određenoj razini
- ***potomci*** (engl. *Descendents*) – svi nasljednici nekog čvora
- ***korijenski čvor*** (engl. *Root Node*) – početni čvor od kojeg krećemo
- ***ciljni čvor*** (engl. *Goal Node*) – završni čvor do kojeg želimo doći (može ih biti i više)
- ***proširiti čvor*** (engl. *Expanding the Node*) – uključiti u stablo rješenja problema svu djecu određenog čvora
- ***faktor grananja*** (engl. *Branching Factor*) b – broj djece određenog roditeljskog čvora.

Čvor koji smo već uključili, ali nismo još proširili zove se ***otvoreni čvor*** (engl. *Open Node*), a onaj koje smo već proširili zove se ***zatvoreni čvor*** (engl. *Closed Node*).

Formalna definicija prostora pretraživanja je šestorka:

$$(S, A, c, s_0, S_g, h_0) \tag{3-1}$$

gdje su:

S – konačni skup svih mogućih rješenja $\{s_i\}$ kojim gradimo prostor pretraživanja

A – konačni broj akcija $\{a_k\}$ (operatora prijelaza) koje dozvoljavaju prelazak iz jednog čvora (mogućeg rješenja) u drugi

c – cijena prijelaza iz jednog čvora u drugi *c(i,j)* korištenjem neke od akcija iz skupa *A*

s_0 – početno stanje

S_g – skup svih ciljnih stanja $\{s_{gj}\}$ u biti podskup skupa *S* ($Sg \subset S$) – treba biti najmanje jedno ciljno stanje s_g, ali ih može biti i više

h_0 – heuristička funkcija koja je poznata na početku i pomaže u usmjeravanju postupka pretraživanja.

Pogledajmo nekoliko primjera:

Slagalica 3 x 3 (slika 3-2)

Prostor pretraživanja gradimo devetorkama brojeva kojima prikazujemo pojedino stanje slagalice. Za prazno mjesto koristimo 0. Na primjer, početno stanje s_0 je (8, 0, 6, 5, 4, 7, 2, 3, 1) i imamo samo jedno ciljno stanje s_g (0, 1, 2, 3, 4, 5, 6). Prostor pretraživanja može činiti ukupno 181440 različitih kombinacija ($|S| = 9!/2$).

Dozvoljene akcije su pomak lijevo, desno, dolje i gore na mjesto gdje se nalazi 0, s tim da su sve dozvoljene jedino ako je 0 na 5. mjestu u prikazu stanja slagalice. Ako je 0 na rubovima slagalice (na 1., 3. 7. i 9. mjestu), dozvoljena su samo dva poteza lijevo (ili desno) ili gore (ili dolje). Treća je situacija kada je 0 na rubu u sredini (na 2. , 4. , 6. i 8. mjestu) i kada su moguća tri poteza. Na primjer, dvije dozvoljene akcije kada je 0 na 1. mjestu možemo prikazati implikacijama:

(0, x, ?, ?, ?, ?, ?, ?, ?) → (x, 0, ?, ?, ?, ?, ?, ?, ?)

(0, ?, ?, x, ?, ?, ?, ?, ?) → (x, ?, ?, 0, ?, ?, ?, ?, ?)

Ukupno postoje 24 ovakve akcije koje uključuju sve moguće poteze, ovisno o tome gdje se 0 nalazi. U ovom slučaju $|A| = 24$.

Cijena prijelaza *c* je 1 (sve su akcije jednako vrijedne).

Faktor grananja *b* kod ovog zadatka je 2, 3 ili 4 ($b \in \{2,3,4\}$).

O heurističkoj funkciji za ovakav tip zadatka kasnije će biti više govora. Ona pomaže u usmjeravanju postupka pretraživanja.

Problem najkraćeg puta na primjeru otoka Brača (slika 3-8)

Prostor pretraživanja čini 8 naselja otoka Brača: Supetar (S), Donji Humac (A), Nerežišća (B), Splitska (C), Pučišća (D), Pražnice E), Gornji Humac (F) i Sumartin (G), tako da se u svakom čvoru nalazi jedno od slova koja predstavljaju ova naselja.

Početno stanje s_0 je naselje Supetar (S), a ciljno stanje s_g naselje Sumartin (G). Cilj smo dosegli ako se u nekom od novootvorenih čvorova pojavi G.

Dozvoljene akcije definirane su slikom 3-8 koja prikazuje iz kojeg se grada može ići u koji, a možemo ih definirati i tablicom 3-1 koja ujedno daje i cijenu (*c*) prijelaza iz jednog čvora prostora pretraživanja u drugi.

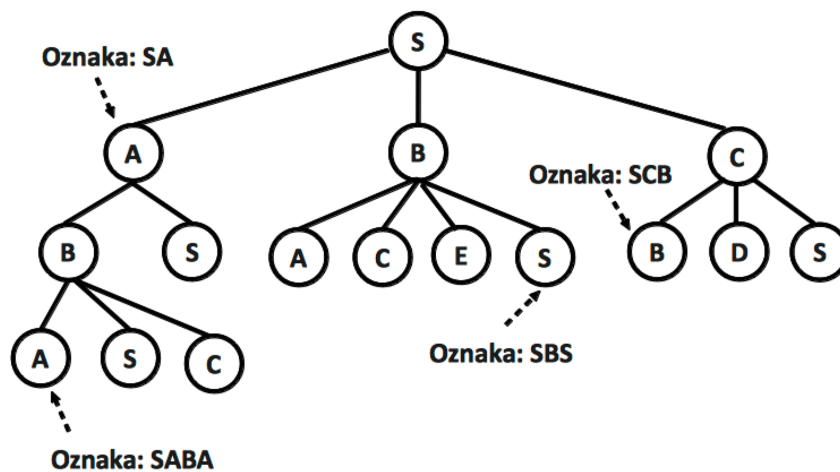
Tablica 3-1. Dozvoljene akcije (operatori prijelaza) u zadatku najkraćeg puta na otoku Braču gdje X znači da ne postoji direktni put između naselja

	Supetar (S)	**Donji Humac (A)**	**Nerežišća (B)**	**Splitska (C)**	**Pučišća (D)**	**Pražnice (E)**	**Gornji Humac (F)**	**Sumartin (G)**
Supetar (S)	0	8	9	7	X	X	X	X
Donji Humac (A)	8	0	2	X	X	X	X	X

Nerežišća (B)	9	2	0	8	X	13	X	X
Splitska (C)	7	X	8	0	22	X	X	X
Pučišća (D)	X	X	X	22	0	11	X	X
Pražnice (E)	X	X	13	X	11	0	4	X
Gornji Humac (F)	X	X	X	X	X	4	0	16
Sumartin (G)	X	X	X	X	X	X	16	0

Faktor grananja i u ovom je zadatku 1, 2 ili 3, a heurističku funkciju kasnije ćemo detaljnije razmatrati.

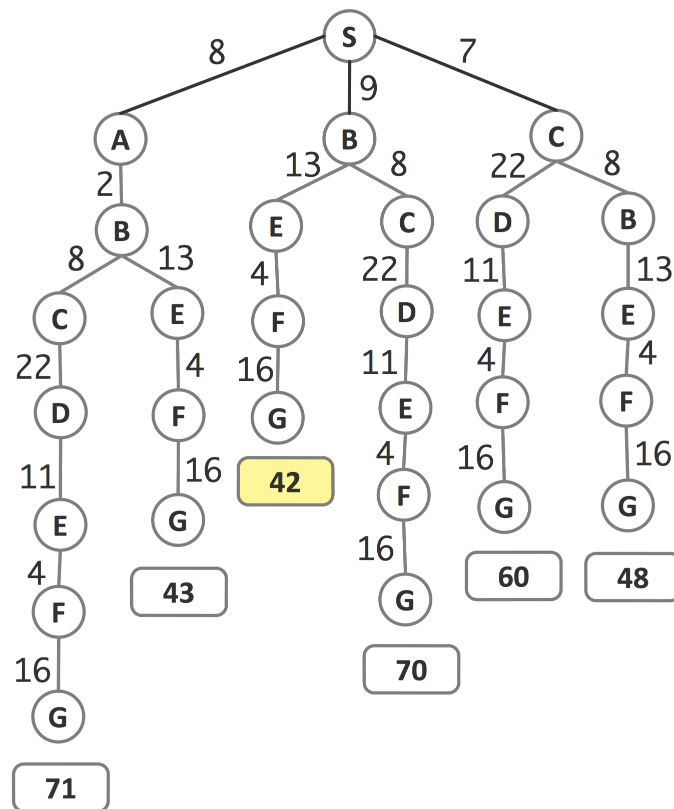
U ovom tipu zadatka nije nam samo cilj doći u konačno, ciljno stanje već i odrediti put do njega. Zbog toga se kod formalnog definiranja ovog zadatka koristi izmijenjena notacija koja u sebi uključuje i prijeđeni put. Slika 3-9 prikazuje dio stabla rješenja problema i način kako se pojedini čvor formalno označava.



Slika 3-9. Dio stabla rješenja problema najkraćeg puta bez udaljenosti s formalnim označavanjem čvorova koji uključuju i informaciju o redoslijedu prolaska čvorova

Početni čvor je *S* i iz njega krećemo. Njegova djeca su čvorovi *A, B* i *C*, ali mi ih označavamo *SA, SB* i *SC* kako bismo sačuvali informaciju o tome da smo već bili u čvoru *S* i iz njega došli u čvorove *A, B, C*. Isti se princip primjenjuje i dalje. Iz čvora *A* možemo ići u čvor *B* ili se vratiti u *S*, pa su njegova djeca *SAB* i *SAS*. Na ovaj način jednostavno detektiramo i to jesmo li se vratili unatrag i započeli petlju. Drugi nasljednik *SAS* kaže da smo se iz *A* vratili natrag u *S*, pa u nastavku nema smisla dalje nastavljati njegovo proširivanje, zato što ćemo se vrtjeti u petlji. Ista je situacija u nastavku pa *SABA* kaže da nam je ovo drugi put u čvoru *A*, znači vrtimo se u petlji. Ovaj zadatak ima 6 mogućih rješenja koje prikazuje slika 3-10.

Od svih rješenja jedno je najbolje zato što je ukupni prijeđeni put najkraći (42 km). Kako se do njega može sustavno doći, opisujemo u sljedećim poglavljima.



Slika 3-10. Sva moguća rješenja zadatka najkraćeg puta

3.2 Izbor prikladnog pravila

Do sada smo naglasili da se postupak rješavanja zadatka sastoji od traženja rješenja u prostoru pretraživanja uz pomoć pravila koja opisuju moguće akcije, prijelaz iz jednog stanja u drugo, ali još ništa nismo rekli o tome kako u svakom pojedinom koraku izabrati najprikladnije pravilo, na koji način graditi stablo rješenja problema.

Moguće akcije kojima se iz jednog stanja prelazi u drugo definirane su skupom A. Najjednostavniji način izbora pravila je sustavni prolazak kroz sva pravila uz odabir onoga kojeg možemo primijeniti u trenutnom stanju. Međutim, ova metoda ima nedostatak ako je broj pravila velik, a to je čest slučaj kod stvarnih primjena.

Postupci pretraživanja baš se i razlikuju po tome koja pravila odabiremo i na koji način nastavljamo graditi stablo rješenja problema. Strategije pretraživanja obično dijelimo na:

- unaprijedno i
- povratno pretraživanje,

te na:

- neinformirano i
- informirano pretraživanje,

o čemu će biti više riječi u nastavku. Kod vrednovanja pojedine strategije pretraživanja postavljamo četiri pitanja:

- Je li se sigurno nalazi rješenje?
- Je li rješenje najbolje?

- Kolika je vremenska složenost?
- Kolika je prostorna složenost?

Vremenska složenost vezana je s brojem razina u kojima se šire čvorovi i zamjenjuju njihovom djecom, a prostorna složenost vezana je s brojem otvorenih čvorova. Ove obje veličine ovise o:

- ***b*** – faktoru grananja koji smo već spominjali (obično se koristi prosječni faktor grananja, ali može i najveći)
- ***m*** – dubini na kojoj se očekuje najbolje rješenje (rješenje s najmanjom cijenom puta)
- ***d*** – maksimalnoj dubini stabla rješenja problema (može biti i beskonačna).

Ako postupak pretraživanja sigurno pronalazi rješenje i to najbolje, te ako uz to ima i malu prostornu i vremensku složenost, onda je on sigurno bolji od onoga koji nam ne garantira da se do rješenja uopće može doći.

3.3 Unaprijedno i povratno pretraživanje

Cilj postupka pretraživanja je otkrivanje puta u prostoru mogućih rješenja od početnog stanja do ciljnog stanja. Postoje dva smjera u kojima se pretraživanje može provesti:

- ***unaprijed*** – od početnog stanja prema ciljnom stanju – ***unaprijedno pretraživanje*** (engl. *Forward Chaining*)
- ***unatrag*** – od ciljnog stanja prema početnom stanju – ***povratno pretraživanje*** (engl. *Backward Chaining*).

Kod unaprijednog pretraživanja testira se lijeva strana pravila koja definira uvjete da bi se dotično pravilo primijenilo, dok se kod povratnog pretraživanja testira desna strana (mogući rezultat). Kao ilustrativni primjer pogledajmo temeljnu strukturu ***ekspertnog sustava***. Radi se o dijagnostičkom ekspertnom sustavu industrijskog procesa. Baza znanja sastoji se od 4 različita tipa ***uzročno-posljedičnih pravila*** (engl. *If-Then Rules*):

1. *Ako <okidač> tada predlažem <hipotezu>.*
   Na osnovu opažanja situacije ili opažanja stanja produkta postavlja se određena hipoteza. Kao primjer možemo uzeti ekspertni sustav za pečenje kolača:
   „Ako je kolač izgorio, tada je peć previše vruća."

2. *<Hipoteza> je uzrokovana <pogreškom>.*
   Razmatramo što je moglo uzrokovati hipotezu:
   „Peć je previše vruća ako je zakazalo hlađenje."

3. *<Greška> se potvrđuje <testom>.*
   „Hlađenje je zakazalo ako je ventil na ON, priključak vode je hladan i pumpa ne vibrira."

4. *Ako je <greška> tada poduzmi <akciju>.*
   „Ako je hlađenje zakazalo i pumpa ne vibrira, tada je potrebno zamijeniti pumpu."

Unaprijedno zaključivanje je pretraživanje od opažene posljedice (okidača) preko hipoteze, greške, prema akciji. Npr. početak traženja iniciran je činjenicom „*Kolač je izgorio.*". Kod povratnog zaključivanja krećemo unatrag, npr. od testa da pumpa ne vibrira i idemo unatrag dok ne dođemo do moguće posljedice, a jedna od njih će biti da kolač može izgorjeti.

Kod problema najkraćeg puta unaprijedno pretraživanje je traženje puta od Supetra do Sumartina, a povratno pretraživanje od Sumartina do Supetra.

Ponekad je jedan od smjerova pretraživanja efikasniji od drugoga, odnosno s njim brže dolazimo do rezultata, pa obično sve metode heurističkog traženja imaju mogućnost i unaprijednog i povratnog

pretraživanja. Pitanje je može li se pretpostaviti na osnovu značajki zadatka koje je pretraživanje efikasnije. Neki od faktora koji na to utječu su:

- *Ima li više početnih ili ciljnih stanja?* – Obično idemo od onog stanja koje ima manji broj varijacija.
- *U kojem je smjeru faktor grananja veći?* – Bolji je onaj smjer u kojem je faktor grananja manji.
- Ako moramo objašnjavati postupke, tada je bolji onaj smjer kod kojeg imamo bolje poklapanje sa slijedom ljudskih misli.

3.4 Neinformirano i informirano pretraživanje

Druga podjela postupaka pretraživanja je vezana uz informacije s kojima raspolažemo kada širimo čvorove i gradimo stablo rješenja problema.

Ako pri odabiru strategije pretraživanja ne koristimo nikakve informacije, onda se takvo pretraživanje naziva ***neinformirano pretraživanje*** (engl. *Non-informed Search*), ***slijepo pretraživanje*** (engl. *Blind Search*) ili ***pretraživanje grubom silom*** (engl. *Brute Force Search*).

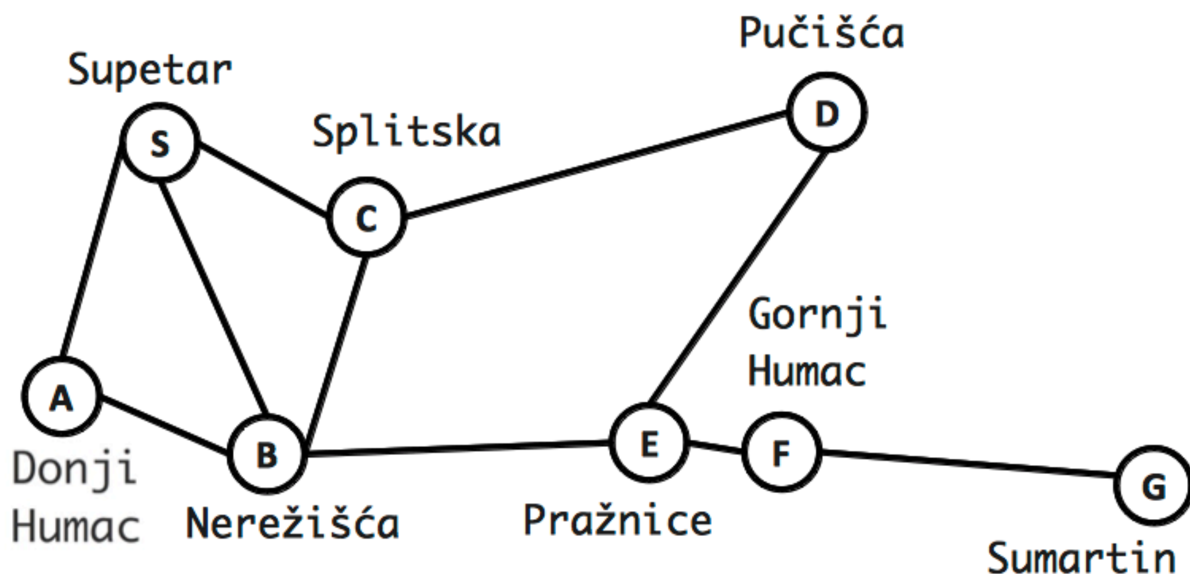
S druge strane, ako kod odabira načina izgradnje stabla problema koristimo neke informacije koje mogu biti pouzdane, determinističke, ali i nepouzdane, heurističke, tada se pretraživanje zove ***informirano pretraživanje*** (engl. *Informed Search*) ili ***usmjereno pretraživanje*** (engl. *Directed Search*). Sigurno je da su ovakve strategije pretraživanja efikasnije.

3.5 Neinformirano ili slijepo pretraživanje

Neinformirano ili slijepo pretraživanje zahtijeva poznavanje:

- početnog stanja
- dopuštenih akcija prijelaza iz jednog stanja u drugo i
- testa kojim se provjerava jesmo li došli u ciljno stanje.

Primjer postavljanja zadatka najkraćeg puta kod neinformiranog pretraživanja prikazuje slika 3-11:



Slika 3-11. Cestovna mreža otoka Brača i formalno prikazani problem najkraćeg puta na pojednostavljenoj mreži s 8 naselja otoka Brača za neinformirano (slijepo) pretraživanje

Jedina dostupna informacija je koja su naselja direktno povezana, pa tablicu dozvoljenih akcija prikazuje tablica 3-2.

Kod slijepog pretraživanja najbolje rješenje je ono koje se pronađe u najmanjem broju koraka (najmanjem broju razina stabla rješenja problema). Na primjeru svih mogućih rješenja problema najkraćeg puta na slici 3-10 vidimo da se prvo rješenje pronalazi na 5. razini, nakon pet širenja čvorova. Pri tome je put opisan sa *SBEFG*. Za njega je ujedno i put najkraći, ali kod slijepog traženja informacija o prijeđenom putu nije nam važna, zato što nije poznata. U ovom se primjeru poklopilo da je put s najmanjim brojem koraka i najkraći, ali to nije opće pravilo. Najgore je rješenje prvo rješenje na lijevoj strani kojem treba 8 razina, a put je *SABCDEFG*.

Strategije neinformiranog ili slijepog pretraživanja razlikuju se po tome kako se prilikom širenja čvora odlučuje kojim putem ići u sljedećoj razini širenja. Dva su pristupa:

- ***sustavni pristup*** kod kojega se na unaprijed dogovoreni način odlučuje na koji način graditi stablo rješenja zadatka (pretraživanje u širinu, pretraživanje u dubinu, pretraživanje ograničeno po dubini i iterativno pretraživanje u dubinu), te
- ***slučajni pristup*** kod kojeg nema nikakvih pravila kako u nekom čvoru odabrati put kojim ćemo nastaviti graditi stablo rješenja zadatka (ne-determinističko slijepo pretraživanje).

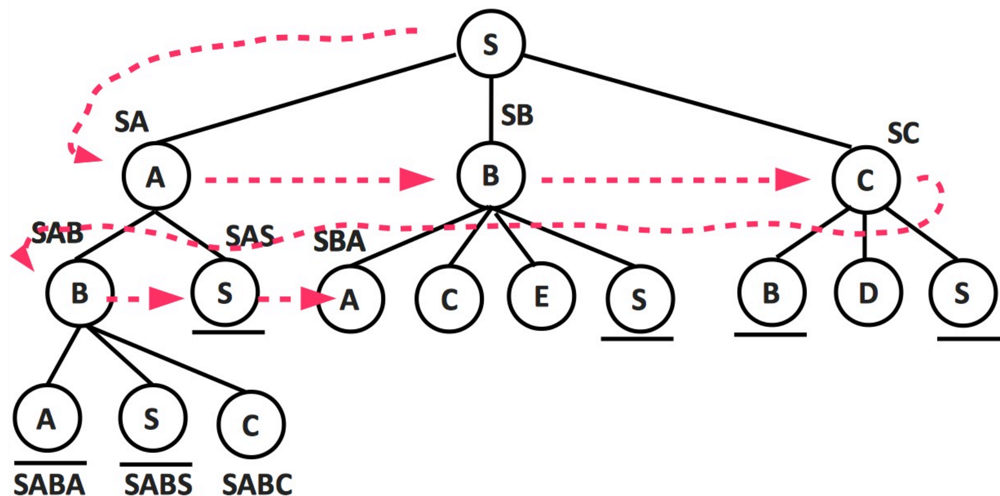
Tablica 3-2. Dozvoljene akcije (operatori prijelaza) u zadatku najkraćeg puta na otoku Braču za neinformirano (slijepo) pretraživanje gdje D znači da postoji, a X da ne postoji direktni put između naselja

	Supetar (S)	**Donji Humac (A)**	**Nerežišća (B)**	**Splitska (C)**	**Pučišća (D)**	**Pražnice (E)**	**Gornji Humac (F)**	**Sumartin (G)**
Supetar (S)	0	D	D	D	X	X	X	X
Donji Humac (A)	D	0	D	X	X	X	X	X
Nerežišća (B)	D	D	0	D	X	D	X	X
Splitska (C)	D	X	D	0	D	X	X	X
Pučišća (D)	X	X	X	D	0	D	X	X
Pražnice (E)	X	X	D	X	D	0	D	X
Gornji Humac (F)	X	X	X	X	X	D	0	D
Sumartin (G)	X	X	X	X	X	X	D	0

Svi ovi algoritmi mogu biti ***jednodirekcijski***, kada se ide od početnog stanja prema ciljnom kod unaprijednog pretraživanja, ili od ciljnog prema početnom kod povratnog pretraživanja, ali i ***bidirekcijski*** kada se istovremeno kreće unaprijed i unatrag, pa je zadatak riješen kada se dva smjera pretraživanja poklope u istom stanju (istom čvoru stabla rješenja problema).

3.5.1 Pretraživanje u širinu

Pretraživanje u širinu (engl. *BFS – Breadth First Search*) je jedan od temeljnih sustavnih postupaka pretraživanja kod kojeg se na svakoj razini šire svi čvorovi koji se mogu otvoriti. Slika 3-12 prikazuje nekoliko koraka u formiranju stabla rješenja postupkom pretraživanja u širinu za zadatak najkraćeg puta sa slike 3-11.



Slika 3-12. Nekoliko koraka u izgradnji stabla rješenja problema kod pretraživanja u širinu

Prema slici 3-10 vidimo da će se pretraživanjem u širinu pronaći najbolje rješenje, zato što će se na 5. razini prvi put doći do cilja (naselja *G*), pa će test zaustaviti daljnje traženje.

U ovom ćemo dijelu algoritam pretraživanja prikazati pseudokodom (algoritam 3-1), a u nastavku će se za sve ove algoritme dati i kod u programskim jezicima koji se u umjetnoj inteligenciji najčešće koriste.

Algoritam 3-1. Pseudokod algoritma pretraživanja u širinu

```
1.  Pohrani početni čvor u niz.
2.  DOK niz nije prazan i nije dosegnut cilj:
3.  Skupi svu djecu prvog člana niza.
4.  Odbaci djecu koja zatvaraju petlju.
5.  Provjeri je li dosegnut cilj.
6.  Izbriši prvi član niza, a djecu koja su ostala dodaj NA KRAJ NIZA.
7.  Ako smo došli do cilja, vrati USPJEH, ako nismo, vrati POGREŠKU.
```

Kod primjera najkraćeg puta simbolički zapis nekoliko koraka sustavne izgradnje stabla rješenja problema je:

(S):: Ukloni S i zamijeni ga njegovom djecom SA, SB, SC.

(SA, SB, SC) :: Ukloni SA i zamijeni ga njegovom djecom SAB i SAS. SAS odbaci zato što se radi o petlji (vratili smo se u S).

(SB, SC, SAB):: Ukloni SB i zamijeni ga njegovom djecom SBA, SBC, SBE, SBS. SBS odbaci zato što se radi o petlji.

(SC, SAB, SBA, SBC, SBE):: Ukloni SC i zamijeni ga njegovom djecom SCB, SCD, SCS. SCS odbaci zato što se radi o petlji.

(SAB, SBA, SBC, SBE, SCB, SCD):: Ukloni SAB i zamijeni ga njegovom djecom SABA, SABS, SABC. SABA i SABS odbaci zato što se radi o petljama.

(SBA, SBC, SBE, SCB, SCD, SABC):: itd.

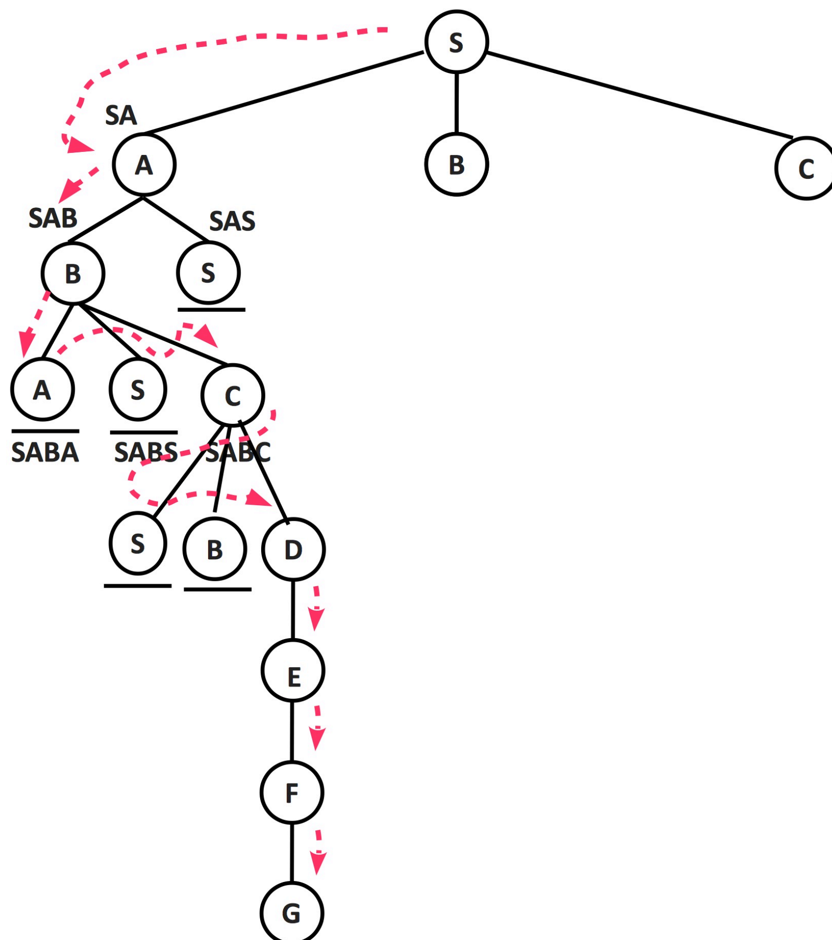
Na petoj razini jedan od nasljednika čvora *SBEF* bit će *SBEFG*. Test koji provjerava samo zadnje slovo simboličkog zapisa ustanovit će da smo došli do cilja – naselja G, te vratiti informaciju o uspjehu.

Pretraživanje u širinu je:

- ***potpuno*** – uvijek pronalazi rješenje za konačni faktor grananja *b* i to najbolje rješenje – rješenje do kojeg se dolazi u najmanjem broju koraka (kod zadatka najkraćeg puta simbolički zapis ima najmanje slova *SBEFG*)
- ***ima veliku vremensku i prostornu složenost*** $O(b^{m+1})$ gdje je *b* faktor grananja, a *m* dubina na kojoj se očekuje najbolje rješenje. Na primjer, za $b = 4$ i $m = 16$, uz pretpostavku da nam je za pohranu svakog stanja potrebno 100 B (Byta), ukupno stablo rješenja trebat će malo više od 1,7 TB.

3.5.2 Pretraživanje u dubinu

Pretraživanje u dubinu (engl. *DFS - Depth First Search*) je drugi temeljni sustavni postupak slijepog pretraživanja kod kojeg se razvija jedna grana dok god se ili ne dođe do rješenja ili dosegne limit dubine pretraživanja (d). Slika 3-13 prikazuje nekoliko koraka u formiranju stabla rješenja postupkom pretraživanja u dubinu za zadatak najkraćeg puta sa slike 3-11.



Slika 3-13. Nekoliko koraka u izgradnji stabla rješenja problema kod pretraživanja u dubinu

Pseudoalgoritam pretraživanja u dubinu prikazuje algoritam 3-2.

Algoritam 3-2. Pseudokod algoritma pretraživanja u dubinu

```
1. Pohrani početni čvor u niz.
2. DOK niz nije prazan i nije dosegnut cilj:
3. Skupi svu djecu prvog člana niza.
4. Odbaci djecu koja zatvaraju petlju.
5. Provjeri je li dosegnut cilj.
6. Izbriši prvi član niza, a djecu koja su ostala dodaju NA POČETAK NIZA.
7. Ako smo došli do cilja vrati USPJEH, ako nismo vrati POGREŠKU.
```

Razlika u odnosu na pretraživanje u širinu je samo u tome što se djeca dodaju na početak reda. Kod primjera najkraćeg puta simbolički zapis nekoliko koraka sustavne izgradnja stabla rješenja problema je:

(S):: Ukloni S i zamijeni ga njegovom djecom SA, SB, SC.

(SA, SB, SC):: Ukloni SA i zamijeni ga njegovom djecom SAB i SAS. SAS odbaci zato što se radi o petlji (vratili smo se u S).

(SAB, SB, SC):: Ukloni SAB i zamijeni ga njegovom djecom SABA, SABS, SABC. SABA i SABS odbaci zato što se radi o petljama.

(SABC, SB, SC):: Ukloni SABC i zamijeni ga njegovom djecom SABCS, SABCB, SABCD. SABCS i SABCB odbaci zato što se radi o petljama.

(SABCD, SB, SC):: itd.

Postupak pretraživanja u dubinu do prvog rješenja dolazi tek na osmoj razini *SABCDEFG*. Postupak je pronašao rješenje, ali nije pronašao najbolje rješenje (s najmanjim brojem koraka), a osim toga za pretpostaviti je da će njegova prostorna složenost biti puno manja.

Pretraživanje u dubinu:

- ***rješenje pronalazi samo u konačnim mrežama***, i to ne najbolje rješenje, a
- ***vremenska složenost mu je jednaka kao kod pretraživanja u širinu*** $O(b^m)$, ali mu je zato prostorna složenost manja, u najboljem slučaju jednaka dubini u kojoj se očekuje najbolje rješenje pomnoženog s faktorom grananja $O(b \cdot m)$.

3.5.3 Nedeterminističko slijepo pretraživanje

Nedeterminističko slijepo pretraživanje (engl. *Non-deterministic Blind Search*) u svakom koraku širenja čvora put kojim će se nastaviti odabire slučajnim izborom (npr. bacanjem kocke). U pseudoalgoritmu postupka razlika je jedino u tome što se djeca dodaju bilo gdje u nizu slučajnim odabirom:

Algoritam 3-3. Pseudokod algoritma nedeterminističkog slijepog pretraživanja

```
1. Pohrani početni čvor u niz.
2. DOK niz nije prazan i nije dosegnut cilj:
3. Skupi svu djecu prvog člana niza.
4. Odbaci djecu koja zatvaraju petlju.
5. Provjeri je li dosegnut cilj.
6. Izbriši prvi član reda, a djecu koja su ostala dodaj BILO GDJE U NIZU SLUČAJNIM
   ODABIROM.
7. Ako smo došli do cilja, vrati USPJEH, ako nismo, vrati POGREŠKU.
```

Nasljednici čvora koji se uklanja dodaju se na slučajno odabrano mjesto. Postupak će doći do rješenja u konačnim mrežama, ako napravimo dovoljan broj pokušaja. Ako imamo sreće, možda će biti i najbolje rješenje, ali se ništa ne može garantirati, kao niti odrediti vremenska i prostorna složenost. U najgorem slučaju bit će potrebno izgraditi cijelo stablo rješenja problema pa će i vremenska i prostorna složenost biti proporcionalna s $O(b^d)$.

Ovaj je postupak na neki način vezan s poznatim ***beskonačnim majmunskim teoremom*** (engl. *Infinity Monkey Theorem*) koji kaže da će majmun slučajno udarajući po tipkama pisaće mašine sigurno napisati bilo koji zadani tekst (na primjer sva djela Shakespearea) ako ima dovoljno vremena. U pojmu dovoljno vremena misli se na jako puno vremena. Pogledajmo za primjer koliko bi majmunu trebalo vremena da napiše riječ *„banana"*. Pisaća mašina ima 50-ak tipki, slova i interpunkcije, pa bi vjerojatnost da će majmun slučajnim udaranjem po tipkama napisati riječ *„banana"* koja predstavlja blok od 6 slova bila $p = (1/50)^6 = 1/15\ 625\ 000\ 000$ (pretpostavljamo da je svaki udarac nezavisan događaj). Vjerojatnost da majmun u bloku od 6 znakova ne napiše riječ *„banana"* je $1-p = 1 - (1/50)^6$. Pretpostavimo sada da je majmun napisao *n* blokova od 6 znakova. Kako je svaki blok nezavisan, vjerojatnost da se nakon *n* blokova ne napiše riječ *„banana"* je $P_n = (1 - (1/50)^6)^n$. Kako *n* raste tako ova vjerojatnost pada. Za milijun blokova je 0,999, a za 10 milijardi 0,53, a ako *n* teži u beskonačno, vjerojatnost da ne napiše riječ *„banana"* je 0, što znači da će majmun sigurno, s vjerojatnošću 100%, nakon beskonačno pokušaja napisati riječ *„banana"*.

3.5.4 Pretraživanje ograničeno po dubini

Pretraživanje ograničeno po dubini (engl. *DLS – Depth Limited Search*) je pretraživanje u dubinu kod kojeg je limitirana dubina do koje se pretražuje. Dubina *n* do koje se ide obično je pretpostavljena dubina *m* na kojoj bi se trebalo pronaći najbolje rješenje. Naglasak je na ***pretpostavljena***. Ako je pretpostavka kriva, rješenje se neće pronaći. Algoritam pretraživanja ograničenog po dubini prikazuje algoritam 3-4:

Algoritam 3-4. Pseudokod algoritma pretraživanja ograničenog u dubinu

```
1. Pohrani početni čvor u niz.
2. DOK niz nije prazan i nije dosegnut cilj:
3. Ako je DUBINA PRETRAŽIVANJA MANJA OD n, skupi svu djecu prvog člana niza.
4. Odbaci djecu koja zatvaraju petlju.
5. Provjeri je li dosegnut cilj.
6. Izbriši prvi član reda, a djecu koja su ostala dodaj NA POČETAK NIZA.
7. Ako smo došli do cilja, vrati USPJEH, ako nismo, vrati POGREŠKU.
```

Pretraživanje ograničeno po dubini je:

- ***potpuno*** ako je pretpostavljena dubina *n* veća ili jednaka dubini *m* na kojoj se nalazi najbolje rješenje i tom slučaju pronalazi se i najbolje rješenje (rješenje do kojeg se dolazi u najmanjem broju koraka), a
- ***prostorna složenost*** je $O(b \cdot n)$ gdje je *n* pretpostavljena maksimalna dubina do koje idemo, dok je ***vremenska složenost*** $O(b^n)$.

3.5.5 Iterativno pretraživanje u dubinu

Iterativno pretraživanje u dubinu (engl. *IDS – Iterative Deepening Search*) razlikuje se od DLS-a samo po tome što se odredi početna pretpostavljena dubina *n*, pa ako se na njoj ne pronađe rješenje, dubina se poveća za 1 (n = n+1), sve dok se rješenje ne pronađe. Pseudokod algoritma prikazuje algoritam 3-5.

Algoritam 3-5. Pseudokod algoritma iterativnog pretraživanja u dubinu

```
1. Pohrani početni čvor u niz.
2. DOK niz nije prazan i nije dosegnut cilj:
3. Kreni od početne dubine m.
4. Ako je DUBINA PRETRAŽIVANJA MANJA OD n, skupi svu djecu prvog člana niza.
5. Odbaci djecu koja zatvaraju petlju.
6. Provjeri je li dosegnut cilj.
7. Izbriši prvi član reda, a djecu koja su ostala dodaj NA POČETAK NIZA.
8. Ako smo došli do cilja, vrati USPJEH, ako nismo, POVEĆAJ DUBINU n = n + 1 pa se
   vrati na vrati korak 5.
9. Ako je dosegnuta najveća dubina d, a nismo došli do cilja, vrati POGREŠKU.
```

Iterativno pretraživanje u dubinu je:

- ***potpuno*** – uvijek pronalazi rješenje i to najbolje rješenje (rješenje do kojeg se dolazi u najmanjem broju koraka)
- ***prostorna složenost*** je $O(b \cdot m)$ gdje je m dubina na kojoj smo pronašli rješenje (ne početna dubina), dok je vremenska složenost $O(b^m)$.

Ovo pretraživanje kombinira dobre osobine i pretraživanja u dubinu i pretraživanja u širinu. Iterativno pretraživanje po dubini sigurno pronalazi najbolje rješenje ako krenemo od početne dubine $n = 1$ i onda je postupno podižemo na $n = 2, 3, \ldots$ sve do m na kojoj se nalazi najbolje rješenje.

3.5.6 Rješavanje problema labirinta slijepim pretraživanjem

Slijepo pretraživanje je posebno značajno kod rješavanja zadataka labirinta, koje u stvarnom svijetu vodi prema autonomnoj navigaciji vozila u nepoznatom okružju. Zamislimo da uđemo u stvarni vegetacijski labirint sa slike 3-14 kod kojeg je vegetacija toliko visoka da vidimo samo nebo. Kod odabira puta jedino što možemo birati jest gdje skrenuti na raskrižju putova.

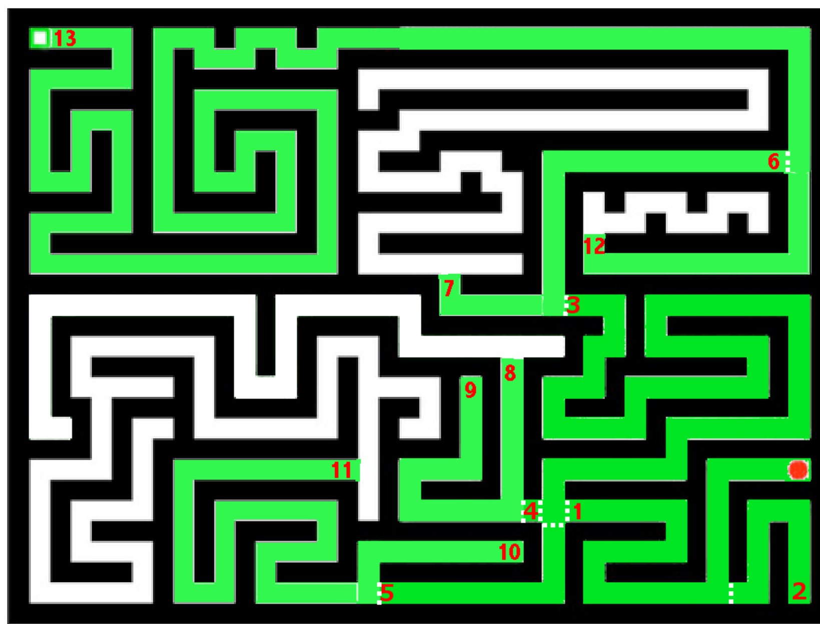


Slika 3-14. Vegetacijski labirint[28]

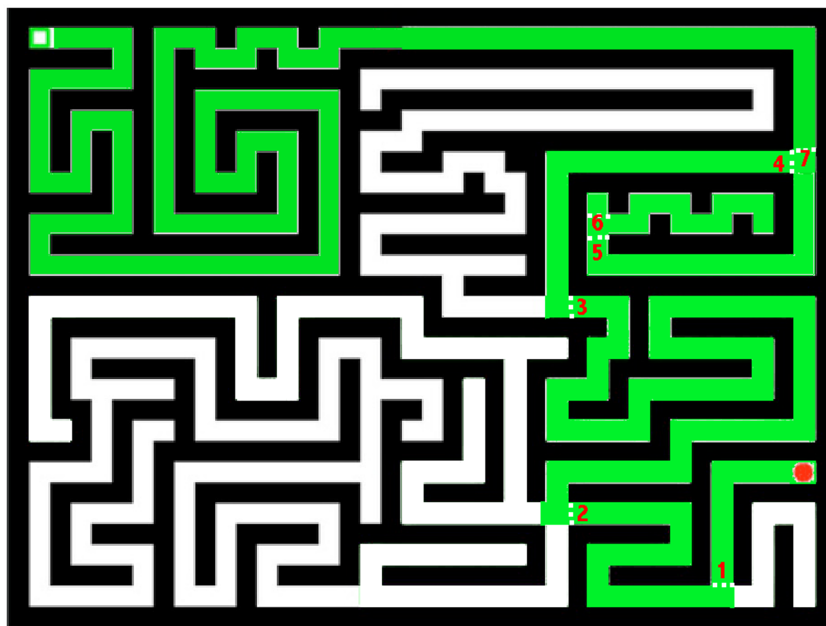
[28] Slika je s https://www.freeimages.com/photo/landscape-labyrinth-1634853 - licenca Public Domain

Pretpostavimo li da je shema labirinta prikazana slikom 3-3, što bi u tom slučaju bilo pretraživanje u širinu, a što pretraživanje u dubinu?

Pretraživanje u širinu bilo bi na svakom raskrižju krenuti na primjer desnim putom do sljedećeg raskrižja. Ako nismo došli do izlaza, vraćamo se natrag i biramo drugi put. Primjer širinskog pretraživanja kod kojeg je otvoreno ukupno 13 grana dok se nije došlo do cilja prikazuje slika 3-15.



Slika 3-15. Širinsko pretraživanje kod rješavanja problema labirinta. U prvom koraku otvore se dva puta – 1 i 2. Kako izlaz nije pronađen, otvara se drugi korak s tri puta – 3, 4 i 5 itd. Na četvrtoj razini grane 3 otvorila su se dva puta 12 i 13 od kojih je 13 doveo do izlaza. Širinsko pretraživanje sigurno će pronaći izlaz iz labirinta, ali ćemo se stvarno našetati (velika prostorna složenost).



Slika 3-16. Dubinsko pretraživanja kod rješavanja problema labirinta. Na svakom je skretanju korišteno „pravilo desnog skretanja" koje kaže da se na svakom skretanju uvijek skrene desno dok se ne dođe do kraja. Nakon toga se vraćamo samo za jedan korak i idemo sljedećim putom ponovo koristeći „pravilo desnog skretanja". Nakon petog dubinskog koraka došli smo do dviju grana koje ne vode nigdje, pa smo se morali vratiti na 4. razinu i krenuti drugim mogućim putom koji je doveo do izlaza.

Kod ***dubinskog pretraživanja*** prikazanog na slici 3-16 korišteno je „***pravilo desnog skretanja***" – na svakom skretanju skrenuti desno, a ako nismo tom granom došli do kraja, vratiti se na prvo prethodno raskrižje i ponoviti postupak novim putom koji nismo prošli. Nakon petog dubinskog koraka došli smo do dviju grana koje ne vode nigdje, pa smo se morali vratiti na 4. razinu i krenuti drugim mogućim putom koji će možda dovesti do izlaza.

3.6 Informirano ili usmjereno pretraživanje

Kod ***informiranog pretraživanja*** (engl. *Informed Search*), ili kako se ponekad naziva ***usmjerenog pretraživanja*** (engl. *Directed Search*), osim početnog stanja, dopuštenih akcija prijelaza iz jednog stanja u drugo, i testa kojim se provjerava jesmo li došli u ciljno stanje, raspolažemo i s određenim informacijama o prostoru pretraživanja i stanju u kojem se trenutno nalazimo.

Na primjeru najkraćeg puta na Braču kod neinformiranog pretraživanja nismo ništa znali o udaljenostima između naselja (slika 3-11 i tablica 3-2), dok kod informiranog pretraživanja koristimo i informacije o udaljenostima između pojedinih naselja (slika 3-8 i tablica 3-1). Ovo su ***pouzdane*** ili ***determinističke informacije*** koje prema formalnoj definiciji postupka pretraživanja (3.1) predstavljaju cijenu prijelaza iz jednog čvora u drugi $c(i,j)$. Udaljenost između Supetra i Nerežišća je sigurno 9 km, pa kada smo prešli taj put, prešli smo baš 9 km. Osim ovakvih pouzdanih informacija kod informiranog pretraživanja mogu se koristiti i ***nepouzdane, heurističke informacije***. Iako su nepouzdane, na njima su se razvili brojni postupci pretraživanja koji su uspješno rješavali i kompleksne probleme. ***IBM-ov Deep Blue*** pobijedio je ***Garija Kasparova*** upravo zahvaljujući dobro odabranoj heuristici pomoću koje se značajno smanjio prostor pretraživanja i ubrzao postupak pronalaženja rješenja.

U nastavku se najprije bavimo usmjerenim pretraživanjem temeljenim na determinističkim informacijama koje se uobičajeno naziva ***optimalno pretraživanje***, zatim usmjerenim pretraživanjem temeljenim na heurističkim informacijama koje zovemo ***heurističko pretraživanje*** i na kraju postupcima pretraživanja koji koriste i determinističke i heurističke informacije i koji su, baš zbog toga, možda i najefikasniji.

3.6.1 Optimalno pretraživanje

Kod ***optimalnog pretraživanja*** (engl. *Optimal Search*) u svakom koraku promatramo koliku smo do sada ***„platili cijenu"*** dolaska do tog čvora i nastavljamo dalje onim putem kod kojeg je dosadašnji trošak bio najmanji:

$$f(s) = c, \quad \text{gdje je } c \text{ cijena do sada prijeđenog puta} \qquad (3\text{-}2)$$

Prethodnik svih algoritama optimalnog pretraživanja je slavni ***Dijkstra algoritam***. ***Edsger W. Dijkstra*** (1930. – 2002.) uveo ga je 1956. godine s ciljem pronalaska najkraćih puteva između čvorova otežanih (usmjerenih ili neusmjerenih) grafova. Cilj mu je bio izgraditi:

- potpunu mapu udaljenosti između čvorova grafa, te
- pronaći najkraći put između bilo koja dva čvora.

Razlika između Dijkstra algoritma i algoritama optimalnog pretraživanja je u tome što se kod optimalnog pretraživanja traži samo najkraći put, a ne potpuna mapa udaljenosti između svih čvorova.

Dijkstra algoritam

Pseudokod Dijkstra algoritma prikazuje algoritam 3-6:

Algoritam 3-6. Pseudokod Dijkstra algoritma

```
1. Formiraj skup svih čvorova S.
2. Za svaki čvor s iz skupa S postavi inicijalno:
3. cijena(s) = 0 AKO je v početni čvor,
```

```
4.  cijena(s) = ∞  ZA SVE OSTALE čvorove v
5.  DOK skup S nije prazan i nije dosegnut cilj:
6.  Izberi najbliže čvorove s početnom čvoru i promijeni njegovu cijenu iz ∞  u cijena(s)
    = c(s0,s), gdje je c(s0,s) cijena dolaska od početnog čvora s0 do čvora s.
7.  Za sve susjedne čvorove v čvorovima s koji još nisu posjećeni promijeni cijenu u
    cijena(v) = cijena(s) + c(s,v).
8.  AKO je neki od čvorova a već bio posjećen, promijeni mu cijenu ako je nova cijena
    MANJA.
9.   Ako smo obišli sve čvorove, vrati USPJEH.
```

Pogledajmo na primjeru Brača. Početna tablica kod koje polazimo iz Supetra (*S*) izgledala bi:

S	*A*	*B*	*C*	*D*	*E*	*F*	*G*
0	∞	∞	∞	∞	∞	∞	∞

U sljedećem koraku gledamo gdje možemo doći iz početnog naselja S (Supetra). To su naselja Donji Humac (*A*), Nerežišća (*B*) i Splitska (*C*). Nadopunimo tablicu udaljenostima do njih:

S	*A*	*B*	*C*	*D*	*E*	*F*	*G*
0	∞	∞	∞	∞	∞	∞	∞
	8	*9*	*7*	∞	∞	∞	∞

U nastavku promatramo možemo li do pojedinih naselja doći i brže preko nekih od naselja kroz koja smo prošli. Na primjer do *B* smo došli uz cijenu 9. Ako bismo išli preko *A* do *B*, cijena bi bila 8 + 2 = 11 što je veće od 9. Isto tako do *B* možemo i preko *C*, ali bi cijena također bila veća (7 + 8 = 15), pa zadržavamo cijenu do *B* na 9 koja je najkraća.

Cijela tablica udaljenosti koja daje najkraće udaljenosti od Supetra (*S*) do svih ostalih naselja je:

S	*A*	*B*	*C*	*D*	*E*	*F*	*G*
0	∞	∞	∞	∞	∞	∞	∞
	8	*9*	*7*	∞	∞	∞	∞
				29	*22*	∞	∞
						26	∞
							42

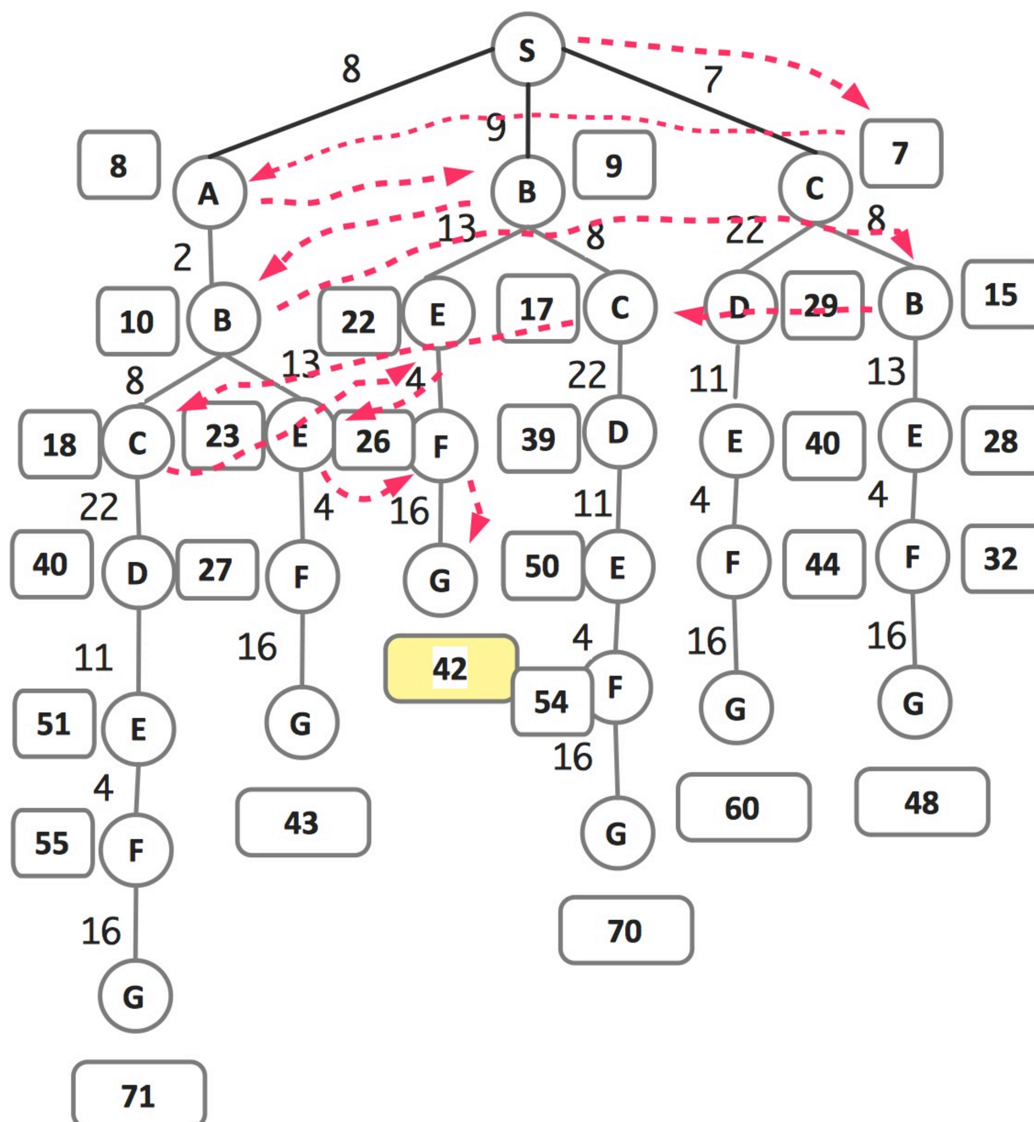
Najmanja udaljenost do Sumartina (*G*) je 42 km što smo već i detektirali na Slici 3-10.

Kako se isti ovaj zadatak rješava postupcima optimalnog pretraživanja, ilustrirat ćemo na istom primjeru s algoritmima:

- pretraživanje jednolikim troškom
- pretraživanje optimalnim jednolikim troškom.

Pretraživanje jednolikim troškom

Pretraživanje jednolikim troškom (engl. *UCS – Uniform Cost Search*) sastoji se od toga da u svakom čvoru krećemo onim putem i širimo onaj čvor za koji je dosadašnja akumulirana cijena bila najmanja. Na primjeru najkraćeg puta na otoku Braču to je ukupan broj prijeđenih kilometara. Slika 3-17 prikazuje stablo rješavanja problema u ovom slučaju.



Slika 3-17. Rješavanje zadatka najkraćeg puta na pojednostavljenoj mreži s 8 naselja otoka Brača za pretraživanje jednolikim troškom

Najbliži grad Supetru (*S*) je Splitska (*C*) na udaljenosti od 7 km, pa idemo tamo. Nakon Splitske možemo ići u Pučišća (*D*) ili u Nerežišće (*B*) za koje je akumulirana cijena 29, odnosno 15. Kako smo već otvorili čvor Donji Humac (*A*) za koji je akumulirana cijena manja i iznosi 8, vraćamo se tamo i gledamo nasljednike. Samo je jedan nasljednik i to Nerežišće (*B*) uz akumuliranu cijenu 10. Opet imamo jedan od početnih čvorova kojem je akumulirana cijena manja i iznosi 9, pa se vraćamo njemu, itd.

Pseudokod algoritma prikazuje algoritam 3-7.

Algoritam 3-7. Pseudokod algoritma pretraživanja jednolikim troškom

```
1. Pohrani početni čvor u niz.
2. DOK niz nije prazan i nije dosegnut cilj:
3. Skupi svu djecu prvog člana niza.
4. Odbaci djecu koja zatvaraju petlju.
5. Provjeri je li dosegnut cilj.
```

```
6. Izbriši prvi član reda, dodaj djecu te nakon toga POREDAJ NIZ PREMA AKUMULIRANOJ
   CIJENI, tako da je na prvom mjestu najmanja akumulirana cijena.
7. Ako smo došli do cilja, vrati USPJEH, ako nismo, vrati POGREŠKU.
```

Kod simboličkog zapisa postupka rješenja svakom putu dodajemo i njegovu težinu:

(0-S) :: Ukloni S i zamijeni ga njegovom djecom SA, SB, SC, koju treba poredati po akumuliranoj cijeni.

(7-SC, 8-SA, 9-SB):: Ukloni SC i zamijeni ga njegovom djecom SCS i SCB i SCD. SAS odbaci zato što se radi o petlji. Cijeli red poredaj po akumuliranoj cijeni

(8-SA, 9-SB, 15-SCB, 29-SCD):: Ukloni SA i zamijeni ga njegovom djecom SAB, SAS. SBS odbaci, a red poredaj po cijeni.

(9-SB, 10-SAB, 15-SCB, 29-SCD):: itd.

Ako je cijena svih putova ista, algoritam se svodi na pretraživanje u širinu. Problem algoritma je što je dosta vremenski i prostorno zahtjevan, a ponekad nije ni optimalan. Može se dogoditi da je akumulirana cijena jednog puta do zadnjeg koraka minimalna, a onda se zadnjim korakom povećava i postaje veća od akumulirane cijene nekog drugog puta. Kao primjer zamislimo da idemo direktno kratkim putom do podnožja planine, a onda moramo napraviti veliku zaobilaznicu, a da smo od početka išli putom zaobilaženja planine, možda bismo u početku prelazili više, ali bi u sumi bilo najmanje. Ovaj problem rješava pretraživanje optimalnim jednolikim troškom.

Pretraživanje optimalnim jednolikim troškom

Pretraživanje optimalnim jednolikim troškom (engl. *OUCS – Optimal Uniform Cost Search*) je mala nadogradnja prethodnog postupka, a sastoji se od toga da ne stanemo kada prvi put dođemo do cilja, već nastavljamo širiti čvorove ako je njihova cijena manja od akumulirane cijene dostignutog cilja. Nadamo se da bismo širenjem nekog od čvorova koji imaju manju akumuliranu cijenu od one koju smo sada postigli pronašli put koji će imati akumuliranu cijenu manju od postignute. Ako ima nade, nastavljamo pretragu.

Pseudokod algoritma možemo ovako napisati:

Algoritam 3-8. Pseudokod algoritma pretraživanja optimalnim jednolikim troškom

```
1. Pohrani početni čvor u niz.
2. DOK niz nije prazan i nije dosegnut cilj i IMA ČVOROVA ČIJA JE AKUMULIRANA CIJENA
   MANJA OD AKUMULIRANE CIJENE POSTIGNUTOG CILJA:
3. Skupi svu djecu prvog člana niza.
4. Odbaci djecu koja zatvaraju petlju.
5. Provjeri je li dosegnut cilj.
6. Izbriši prvi član reda, dodaj djecu te nakon toga POREDAJ NIZ PREMA AKUMULIRANOJ
   CIJENI, tako da je na prvom mjestu najmanja akumulirana cijena.
7. Ako smo došli do cilja, vrati USPJEH, ako nismo, vrati POGREŠKU.
```

Ovaj algoritam sigurno pronalazi optimalno rješenje kao i pretraživanje u širinu, ali je isto tako vremenski i prostorno zahtjevan. Ako je C konačna cijena, a ε prosječna cijena jednog koraka, do cilja nam treba ukupno $(1+C/\varepsilon)$ koraka pa je prostorna i vremenska složenost $O(b^{1+C/\varepsilon})$.

Bi li se metode optimalnog pretraživanja mogle primijeniti kod rješavanja problema labirinta? Odgovor je potvrdan, s tim da bismo tada trebali zapisivati prijeđeni put (npr. broj koraka) u pojedinoj grani između dva raskrižja. Na svakom raskrižju birali bismo put kod kojega je prijeđeni broj koraka do tada manji. Na primjer, za labirint sa slike 3-21 u prvom koraku kraći put dovodi do slijepog kraja (2) pa idemo putem do (1). Sada nam se nude tri puta (3), (4) i (5). Jednoliki trošak kaže kreni najkraćim od njih, a to je prema slici 3-21 sigurno put (4), gdje samo u jednom koraku dolazimo do novog raskrižja. Tako nastavljamo dalje tim putem do (8) zato što je kraći od (9) itd. Sa slike nam je jasno da nam to nije pravi put, ali nas jednoliki trošak vodi tim dijelom dok ponovo ne dođemo do slijepog kraja, a onda ćemo

se ponovo morati vratiti na prethodno raskrižje na kojem smo krenuli u (4), te sada otići na (5) zato što je kraće od (3). Ponovo pogrešan put, ali nas jednoliki trošak tako vodi. Na kraju ćemo doći do cilja, ali ćemo se pošteno našetati i prehodati skoro cijeli labirint.

3.6.2 Heurističko pretraživanje

Kod ***heurističkog pretraživanja*** (engl. *Heuristic Search*) u svakom se koraku koristi pomoćna veličina kojom nastojimo procijeniti koliko smo daleko uspjeli doći do cilja ili se udaljiti od početka. Heuristiku smo već spomenuli u Poglavlju 2. Heurističko pretraživanje je postupak kojim se povećava efikasnost postupka traženja, obično uz žrtvovanje potpunosti.

Uvodi se heuristička evaluacijska funkcija koja svakom stanju s u prostoru rješenja zadatka pridodaje mjeru, broj koji procjenjuje koliko smo se približili cilju ili koliko smo daleko odmakli od početka:

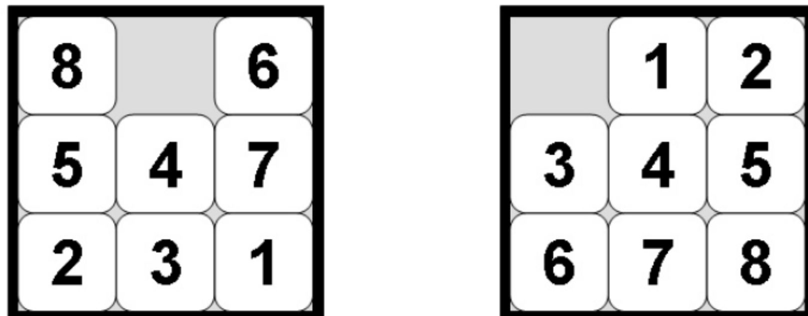
$$f(s) = h \text{ , gdje je } h \text{ procjena cijene do cilja ili procjena cijene od početka} \qquad (3\text{-}3)$$

Naglasak je na procjena, pa h formalno možemo definirati kao preslikavanje sa skupa svih mogućih rješenja S na skup pozitivnih realnih brojeva:

$$h : S \rightarrow \mathbb{R}^{+} \qquad (3\text{-}4)$$

Heurističke funkcije koje procjenjuju udaljenost do cilja obično su efikasnije pa se i više koriste. U tom slučaju kada je cilj g dosegnut $s = g$ i $h(g) = 0$. Heurističku funkciju već smo spomenuli u formalnoj definiciji postupka pretraživanja opisanog jednadžbom *(3-1)* i označili s h_0.

Postoji općenite heuristike koje su primjenjuju kod rješavanje različitih tipova zadataka, ali i specijalizirane heuristike razvijene za potrebe rješavanja zadataka određene, uske domene. Pogledajmo neke primjere. Kod zadataka tipa slagalice sa Slike 3-18 heuristička evaluacijska funkcija koja u trenutnom stanju (na slici lijevo) procjenjuje udaljenost od početka mogla bi biti broj znamenaka dobro postavljenih na svoje mjesto. U našem slučaju to bi bilo $h(s) = 1$, s obzirom na to da se samo broj 4 nalazi na svom mjestu.



Slika 3-18. Trenutno i ciljno stanje kod problema slagalice za koju smo u tekstu opisali različite heurističke funkcije

Ako kao heurističku funkciju uzmemo broj znamenki koje nisu na svom mjestu, onda na neki način procjenjujemo koliko smo daleko do cilja kada bi sve znamenke trebale biti na svom mjestu. U našem primjeru $h(s) = 7$.

Treća heuristička evaluacijska funkcija koja se i najviše koristi, ne samo u ovom problemu već i u problemima labirinta, je ***Manhattan udaljenost***. Ona iskazuje koliko je potrebno napraviti pomaka (horizontalnih i vertikalnih) da bi se sve znamenke dovele na svoje mjesto za slučaj kada bi sve bilo prazno i ne bi smetale druge znamenke. U našem primjeru 1 treba napraviti 3 pomaka, 2 treba napraviti 4 pomaka, itd. Ukupna Manhattan udaljenost je $h(s) = 3 + 4 + 2 + 0 + 2 + 4 + 2 + 4 = 21$.

Kod problema labirinta, uz pretpostavku da znamo koordinate cilja (x_2,y_2) i znamo da se nalazimo na koordinati (x_1,y_1) ***Manhattan udaljenost*** je:

$$h(s) = | x_1 - x_2 | + | y_1 - y_2 | \quad (3\text{-}5)$$

Osim Manhattan udaljenosti koriste se i ***Euklidska udaljenost***:

$$h(s) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (3\text{-}6)$$

Chebyshevova udaljenost:

$$h(s) = max (| x_1 - x_2 | , | y_1 - y_2 | \quad (3\text{-}7)$$

i ***Oktilna udaljenost***:

$$h(s) = max (| x_1 - x_2 | , | y_1 - y_2 |) + (\sqrt{2} - 1) \cdot min (| x_1 - x_2 | , | y_1 - y_2 |) \quad (3\text{-}8)$$

Manhattanska, Euklidska i Chebishevova udaljenost su specijalni slučajevi općenitije udaljenosti koja se zove ***Minkowski udaljenost*** i koristi kao norma normiranih vektorskih prostora. Minkowski udaljenost u dvodimenzionalnom prostoru definira se jednadžbom:

$$h(s) = \sqrt[p]{(x_1 - x_2)^p + (y_1 - y_2)^p} \quad (3\text{-}9)$$

Minkowski udaljenost odgovara Manhattanskoj za $p = 1$, Euklidskoj za $p = 2$ i Chebishevovoj za $p = \infty$.

Euklidska udaljenost je tipična heuristička funkcija koja se koristi kod različitih problema pronalaženja puta, a obično se naziva ***zračna udaljenost*** od pojedinog čvora do ciljnog čvora. Za primjer otoka Brača zračne udaljenosti prikazuje tablica 3-3:

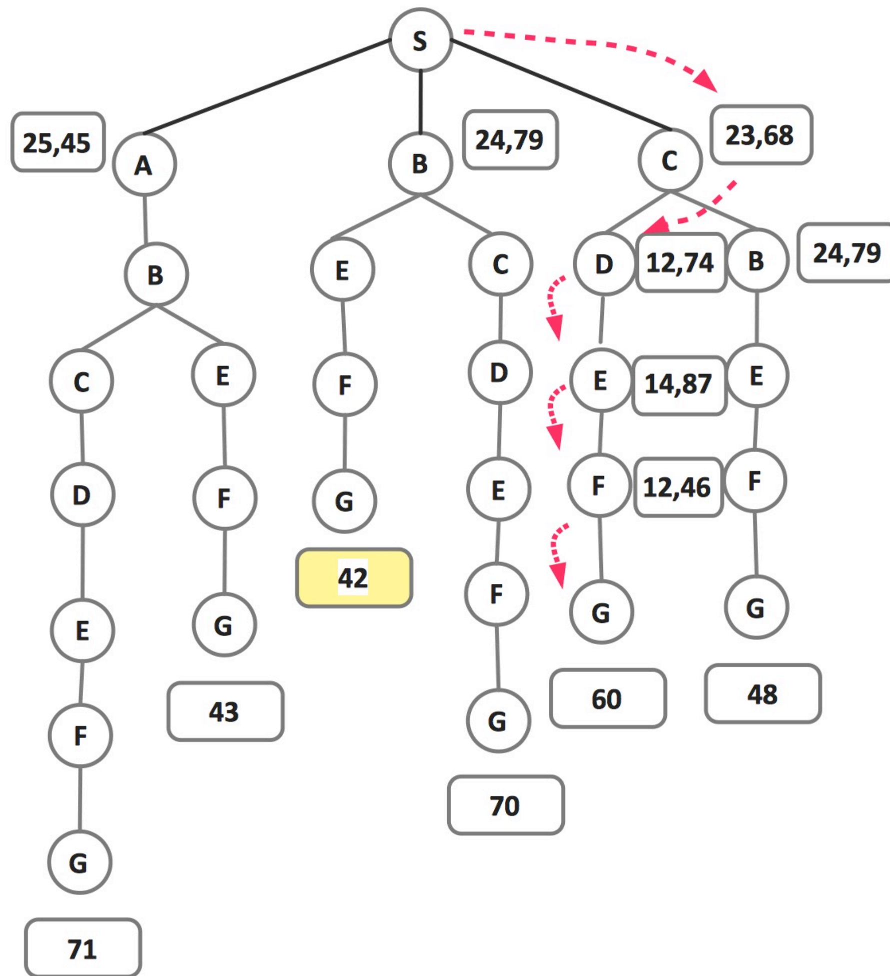
Tablica 3-3. Zračne udaljenosti od pojedinih naselja do cilja – Sumartina (G) u km

Supetar (S)	28,30
Donji Humac (A)	25,45
Nerežišća (B)	24,79
Splitska (C)	23,68
Pučišća (D)	12,74
Pražnice (E)	14,80
Gornji Humac (F)	12,46

Ove ćemo podatke koristiti kod ilustracije različitih algoritama heurističkog pretraživanja. Opisat ćemo ***metodu uspona na brdo***, ***ograničeno širinsko pretraživanje*** i ***najbolje prvo pretraživanje (pohlepno pretraživanje)***.

Metoda uspona na brdo

Metoda uspona na brdo (engl. *Hill Climbing Method*) ili kako se ponekad zove ***metoda poštednog uspona*** slična je pretraživanju u dubinu, samo što se ide samo onim putem koji ima najmanju vrijednost heurističke evaluacijske funkcije. Primjenu metode uspona na vrh na primjeru najkraćeg puta na otoku Braču prikazuje slika 3-19.



Slika 3-19. Rješavanje zadatka najkraćeg puta na Braču metodom uspona na brdo

Lako je uočljivo da je jako brzo došla do cilja, ali rješenje nije optimalno jer do cilja nismo došli najkraćim putem.

Pseudokod algoritma metode uspona na vrh prikazuje algoritam 3-9:

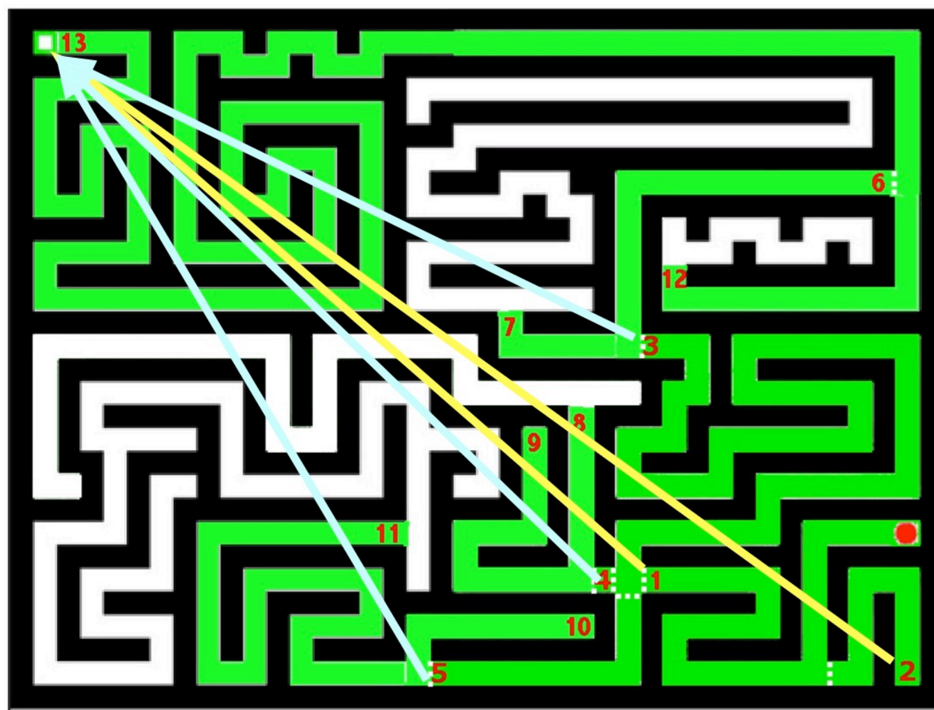
Algoritam 3-9. Pseudokod algoritma uspona na vrh

```
1. Pohrani početni čvor u niz.
2. DOK niz nije prazan i nije dosegnut cilj:
3. Skupi svu djecu prvog člana niza.
4. Odbaci djecu koja zatvaraju petlju.
5. Provjeri  je li dosegnut cilj.
6. Izbriši prvi član reda, dodaj djecu na prvo mjesto, ali ih prije toga poredaj po
   vrijednosti heurističke funkcije.
7. Ako smo došli do cilja, vrati USPJEH, ako nismo, vrati POGREŠKU.
```

Algoritam uspona na vrh inspiriran je penjanjem na planinu u magli. Pretpostavimo da ništa ne vidimo, a želimo se popeti na vrh planine. Ideja je da napravimo korak u svim smjerovima i krenemo onim putom za koji je uspon najveći. Tako postupimo i u drugom koraku, i u trećem koraku sve dok ne dođemo do vrha. Jedini problem jest u tome je li to ujedno i najveći vrh? Možda smo došli do nekog

lokalnog vrha, pa koraci u svim smjerovima vode spuštanju, ili smo došli do hrbata, pa se u dva smjera spuštamo, a u dva ostajemo na istoj visini ili visoravni, što znači da u svim smjerovima više nema napretka. Rješenje je ***skok od sedam milja*** (engl. *Seven-league Boots*) prema dječjoj priči po kojoj se oblačenjem čarobnih čizama može u jednom koraku napraviti sedam milja. Tako i sada, ako smo zapeli na nekom lokanom vrhu trebamo skočiti na sljedeće brdo nadajući se da je njegov vrh upravo onaj koji tražimo.

Kako bismo primijenili metodu uspona na brdo kod problema rješavanja labirinta?



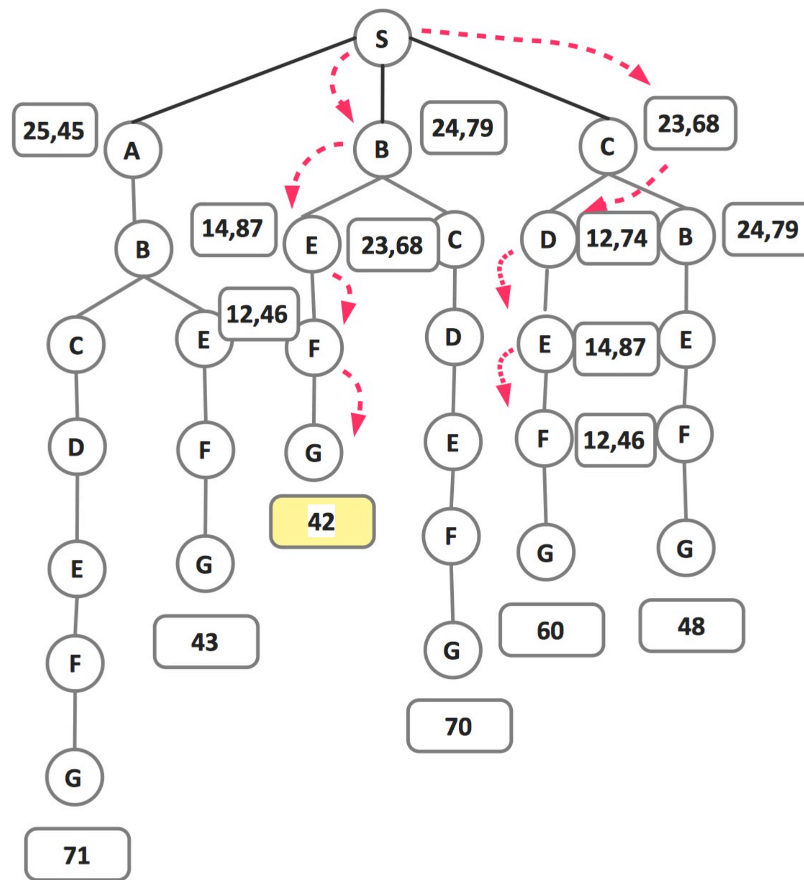
Slika 3-20. Drugi korak rješavanja zadatka labirinta metodom uspona na brdo

Kako se radi o heurističkom informiranom pretraživanju, ono što bismo trebali unaprijed znati jest koordinata izlaza iz labirinta. Trebali bismo znati gdje se izlaz nalazi, ali ne i put do njega. Njega trebamo tek pronaći. Uz poznavanja koordinate izlaza (x_2,y_2) u svakom bi koraku trebao biti poznat i trenutni položaj (x_1,y_1). Put po kojem trebamo nastavili ići određivali bismo na temelju proračuna vrijednosti heurističke funkcije na kraju svakog puta koji smo otvorili. Uvijek nastavljamo dalje onim putem koji ima manju vrijednost heurističke funkcije. Na primjer, ako u prvom koraku prikazanom na slici 3-20 uzmemo Euklidsku udaljenost, onda na drugom križanju krećemo u granu (3) zato što je ona najbliža izlazu po vrijednosti Euklidske udaljenosti.

Metoda uspona na vrh ne garantira pronalazak najboljeg rješenja, pa se može dalje unaprijediti ograničenim širinskim pretraživanjem.

Ograničeno širinsko pretraživanje

Ograničeno širinsko pretraživanje (engl. *BS - Beam Search*) se od uspona na brdo razlikuje jedino u tome što se paralelno spuštamo prema vrijednosti heurističke funkcije po *n* grana na način da odabiremo na svakoj razini onih *n* grana koje imaju najmanju vrijednost heurističke funkcije. Slika 3-21 prikazuje slučaj kada je $n = 2$. Za razliku od metode uspona na brdo, ograničeno širinsko pretraživanje je pronašlo najbolje rješenje.



Slika 3-21. Rješavanje zadatka najkraćeg puta na Braču metodom ograničenog širinskog pretraživanja

Najbolje prvo pretraživanje

Najbolje prvo pretraživanje (engl. *BFS – Best First Search*), ili kako se češće zove ***pohlepno pretraživanje*** (engl. *GS – Greedy Search*), je možda najbolji postupak heurističkog pretraživanja. Slično je metodi uspona na vrh, samo što u svakom koraku širimo bilo koji od otvorenih čvorova koji imaju najmanju vrijednost heurističke funkcije. Kod metode uspona na vrh novi se čvorovi najprije posalože po vrijednosti heurističke funkcije, pa se onda dodaju na početak reda, a kod pohlepnog pretraživanja najprije se dodaju novi članovi, a onda se posloži cijeli red (slično kao kod pretraživanja jednolikim troškom, samo što se niz slaže po vrijednostima heurističke funkcije, a ne po vrijednosti do sada akumulirane cijene).

Pseudokod metode najboljeg prvog pretraživanja prikazuje algoritam 3-8.

Algoritam 3-8. Pseudokod algoritma pretraživanja jednolikim troškom

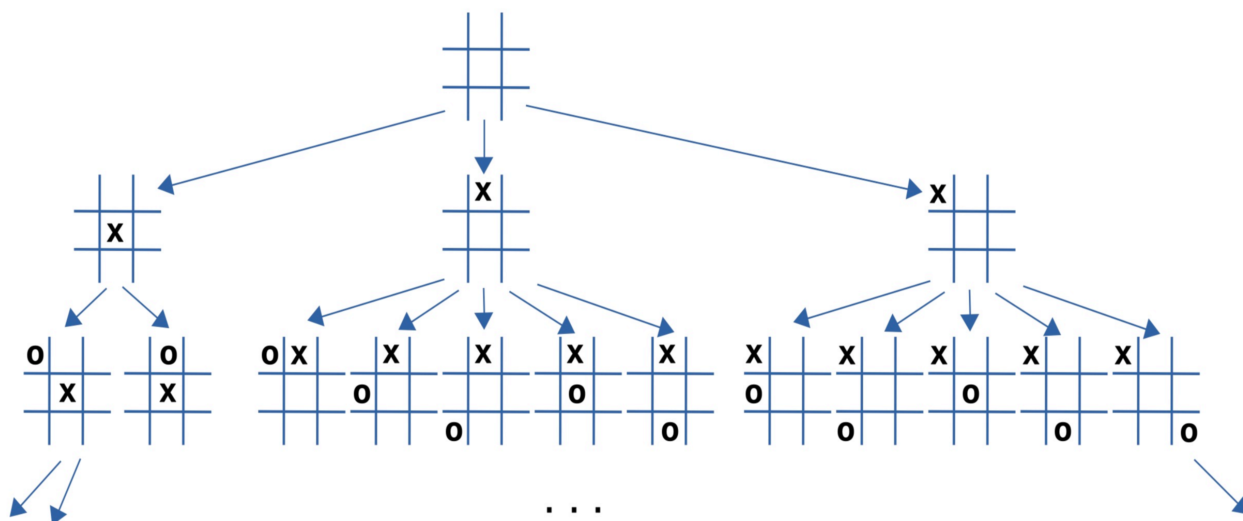
```
1. Pohrani početni čvor u niz.
2. DOK niz nije prazan i nije dosegnut cilj:
3. Skupi svu djecu prvog člana niza.
4. Odbaci djecu koja zatvaraju petlju.
5. Provjeri je li dosegnut cilj.
6. Izbriši prvi član reda, dodaj djecu te nakon toga POREDAJ NIZ PREMA VRIJEDNOSTI
   HEURISTIČKE FUNKCIJE h(s), tako da je na prvom mjestu najmanja vrijednost.
7. Ako smo došli do cilja, vrati USPJEH, ako nismo, vrati POGREŠKU.
```

Vremenska i prostorna složenost pohlepnog pretraživanja ovisi o heurističkoj funkciji, a u najgorem slučaju je eksponencijalna $O(b^d)$, gdje je d maksimalna dubina na kojoj se rješenje pronalazi.

I na kraju spomenimo još dva postupka heurističkog pretraživanja koja se često koriste u problemima dva agenta, prije svega kod igranja igara dva igrača.

Heurističko pretraživanje kod problema dva agenta

Problemi dva agenta su prije svega igre dvaju igrača, na primjer kružić-križić, go, šah, shogi (japanski šah), bridž, poker i slično, kod kojih dva igrača igraju jedan iza drugoga. Kod razvoja stabla rješenja zadatka razine naizmjenično pripadaju jednom, pa drugom igraču. Pogledajmo primjer jednostavne igre križić-kružić. Slika 3-22 prikazuje prva dva poteza u igri kod koje križić igra prvi.



Slika 3-22. Dva koraka igre križić-kružić. Križić počinje prvi i ima zbog simetrije samo tri moguća poteza, staviti križić u sredinu, centralno uz rub ili u kut. Sljedeća razina pripada kružiću i ovisno o tome gdje je postavljen križić, drugi igrač ima na raspolaganju dva ili pet mogućih poteza.

Pitanje je kako izabrati sustavnu tehniku pretraživanja koja bi nas dovela do cilja. Ovdje ćemo ukratko opisati dvije metode koje se često koriste. Jedna je ***MinMax pretraživanje***, a druga ***Monte Carlo pretraživanje stabla***.

MinMax pretraživanje stabla

MinMax pretraživanje (engl. *MM – MinMax Search* ili *MinMax Search*) je pretraživanje kod kojega se pri svakom širenju čvora i određivanju koji ćemo potez odigrati odlučujemo za onaj koji će nam donijeti maksimalni dobitak ili minimalni gubitak. Pretpostavka je da oba igrača igraju optimalno. Igrač koji nastoji postići maksimalni dobitak naziva se „*maximiser*", a onaj koji želi pretrpjeti minimalni gubitak „*minimizer*". Kako se radi o informiranom pretraživanju, svakom stanju (potezu) je potrebno pridijeliti heurističku evaluacijsku funkciju koja procjenjuje koliko smo se približili cilju. U igri križić-kružić to može biti:

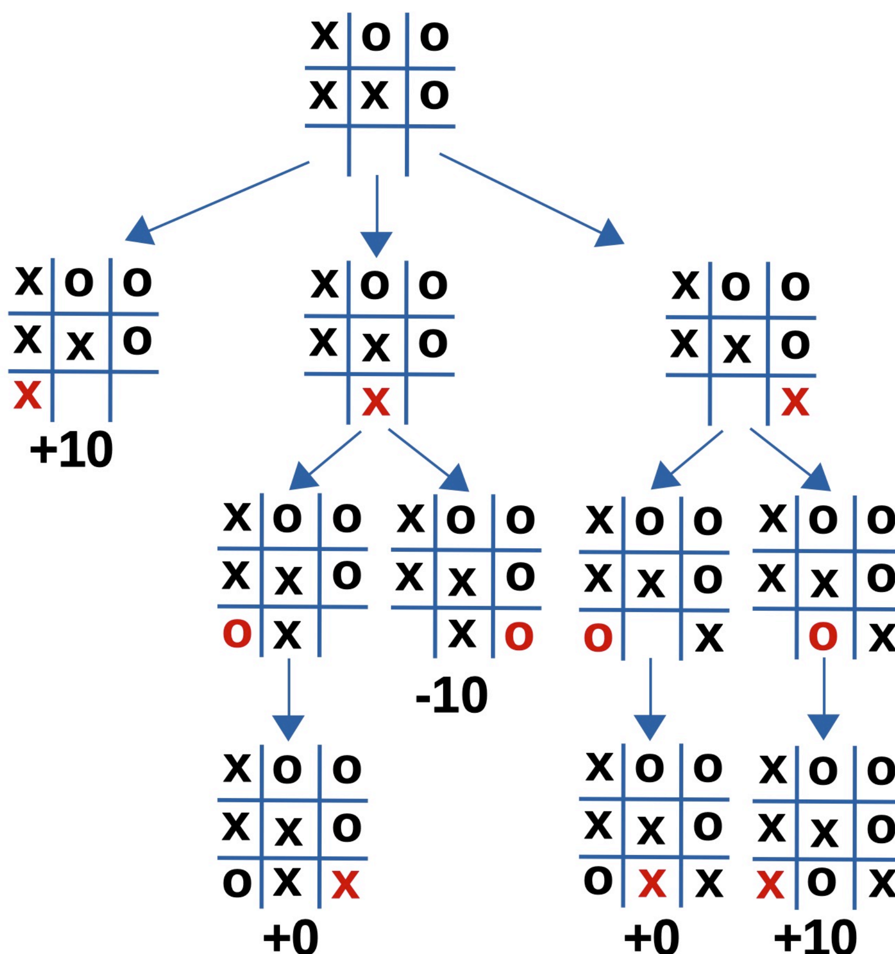
+10 za tri križića u redu (križić je „*maximiser*"),

-10 za tri kružića u redu (kružić je „*minimiser*"),

0 za neriješeni rezultat.

Princip MinMax pretraživanja je „*minimizirati ukupni maksimalni gubitak*", što znači u svakom potezu analizirati sve moguće poteze protivnika dok se ne dođe do kraja u kojem pobjeđuje jedan od igrača ili je situacija neriješena. Kako nije svejedno u kojem se broju koraka dolazi do kraja, dobro je kao

korekcijski faktor uzeti i broj koraka, na način da broj koraka dolaska do cilja oduzimamo od postignute vrijednosti evaluacijske funkcije. Pogledajmo primjer sa slike 3-23.



Slika 3-23. Jedno od mogućih stanja u igri križić-kružić i svi mogući ishodi ovisno o potezima igrača kod MinMax pretraživanja stabla

Na redu je igranje križića i on mora odrediti koji od tri moguća poteza odigrati. Križić simulira sve moguće ishode kojih ima ukupno 5. U dva pobjeđuje križić (+10), u jednome kružić (-10), a postoje dva neriješena ishoda (0). Prvi potez vodi pobjedi nakon jednog koraka, pa je heuristička vrijednost za taj potez 10 – 1 = 9. Drugi potez ima dva ishoda, jedan vodi gubitku pa mu je vrijednost -10, drugi neriješenom rezultatu 0. Treći potez vodi s jedne strane pobjedi, ali u tri koraka, pa mu je vrijednost heurističke funkcije 10 – 3 = 7, a s druge strane neriješenom rezultatu 0. Križić je *„maksimizer"* pa teži maksimalnoj vrijednosti, te je očito da će odigrati zadnji potez koji ima vrijednost heurističke funkcije 9 (i pobijediti).

Problem MinMax pretraživanja je u tome što se moraju odigrati sve igre do kraja kako bi se pronašao najbolji potez. To je lako za jednostavne igre, ali za kompleksnije nije, pa se ponekad MinMax nadopunjuje ograničenjem širenja u dubinu. Promatraju se svi mogući ishodi, ali do neke limitirane dubine koja se isto tako određuje heuristički. Druga zanimljiva nadogradnja MinMax algoritma je tzv. ***alpha–beta obrezivanje*** (engl. *Alpha-Beta Pruning*) kod kojeg se eliminiraju potezi koji su već otkriveni i daju bolji rezultat.

MinMax algoritam ovisi o heurističkoj funkciji, koju ponekad nije jednostavno odrediti, pa su istraživači tražili na koje bi načine mogao uklonili taj nedostatak. Jedno od rješenja je postupak nazvan ***Monte Carlo pretraživanje stabla***.

Monte Carlo pretraživanje stabla

Monte Carlo pretraživanje stabla (engl. *MCTS – Monte Carlo Tree Search*) je univerzalni algoritam određivanja poteza kod igre dvaju igrača, a najviše je uspjeha postigao u kompleksnim igrama kod kojih se MinMax pretraživanje ne bi moglo primijeniti zato što je potencijalni prostor rješenja zadatka prevelik. Primjer je igra ***Go***. MinMax pretraživanje je bilo uspješno dok se nije došlo do igara čija je složenost veća od igre šaha. ***IBM Deep Blue*** koji je pobijedio ***Garija Kasparova*** temeljio se je na MinMax algoritmu, ali se kod igre ***Go*** taj algoritam nije mogao primijeniti pa je stoga razvijena i predložena MCTS metoda. Temeljni dio ovog postupka je usmjeravanje na otvaranje čvorova koji najviše obećavaju, šireći stablo pretraživanja slučajnim uzorkovanjem prostora pretraživanja. Najveći problem ovakvih kompleksnijih zadataka je kombinacijska eksplozija, eksponencijalno širenje broja mogućih akcija, a to je posebno prisutno u naglom rastu broja poteza koje dva igrača mogu odigrati kako igra napreduje.

Ideja Monte Carlo metode temelji se na predviđanju kolika je vjerojatnost da neki potez dovede do pobjede. Ako bismo je mogli odrediti, sigurno bi bilo vjerojatnije da ćemo u igri i pobijediti. Monte Carlo pretraživanje stabla sastoji se od 4 koraka: ***izbor – širenje – simuliranje – ažuriranje*** (engl. *Selection – Expansion – Simulation – Backpropagation*) vrijednosti pojedinih čvorova stabla s ciljem pronalaska konačnog rješenja koje ima najveću vjerojatnost pobjede.

Temelj metode je virtualno igranje igre od početka do kraja slučajnim izborom sljedećeg poteza. Na temelju konačnog rezultata, svakom se čvoru pridodaje vjerojatnost koja ovisi o tome koliko je taj čvor pridonio pobjedi. Naglasak je na *„slučajnom izboru poteza"*, a što se više simulacija napravi, to je i veća šansa pronalaska dobrog poteza. Pogledajmo primjer. Kod igre križić-kružić pretpostavimo da se nalazimo u situaciji prikazanoj na slici 3-23. Početni čvor prikazan na vrhu je naš korijenski čvor *s0*, a čvorovi sljedećeg reda *s01*, *s02* itd. U ovom ćemo primjeru korigirati težine, pa će pobjeda nositi +1, gubitak -1, a neriješeno 0. Sljedeći na redu je križić (x). Slučajnim izborom odigramo prvu virtualnu partiju tako da izaberemo prvi potez i vrlo brzo izgubimo. Kod određivanja vjerojatnosti osim ishoda igre u proračun uzimamo i broj simulacija koje smo napravili startajući iz tog čvora, pa vrijednost ishoda igre podijelimo s tim brojem. Na primjer, u našem slučaju krenuli smo prvim putem i to samo jedanput, pa je $N = 1$. Na kraju smo izgubili i ostvarili $W = -1$, pa su težine čvorova $u_{s0} = -1/1$ i $u_{s01} = -1/1$. Sada odigramo još dva slučajna poteza do kraja (slika 3-24) od kojih jedan daje pobjedu, a jedan neriješeno. Sljedeći je korak unatrag ažuriranje svih težina čvorova koji su prethodili novim simulacijama. Iz početnog čvora odigrali smo tri igre, pa je $N = 3$. Jedanput smo dobili, jedanput izgubili, a jedanput igrali neriješeno, pa je $u_{s0} = (-1+1+0)/3 = 0/3$, ali on i nije toliko bitan s obzirom na to da mi trebamo odlučiti koji potez izabrati na sljedećoj razini. Potez *s01* je ostao na staroj težini $u_{s01} = -1/1$, a kod poteza *s02* odigrali smo dvije igre ($N = 2$), jedanput dobili i jedanput igrali neriješeno, pa je $u_{s02} = (1+0)/2 = 1/2$, što je veće od težine za *s01*, pa bismo se, bez daljnjih simulacija, odlučili za *s02*.

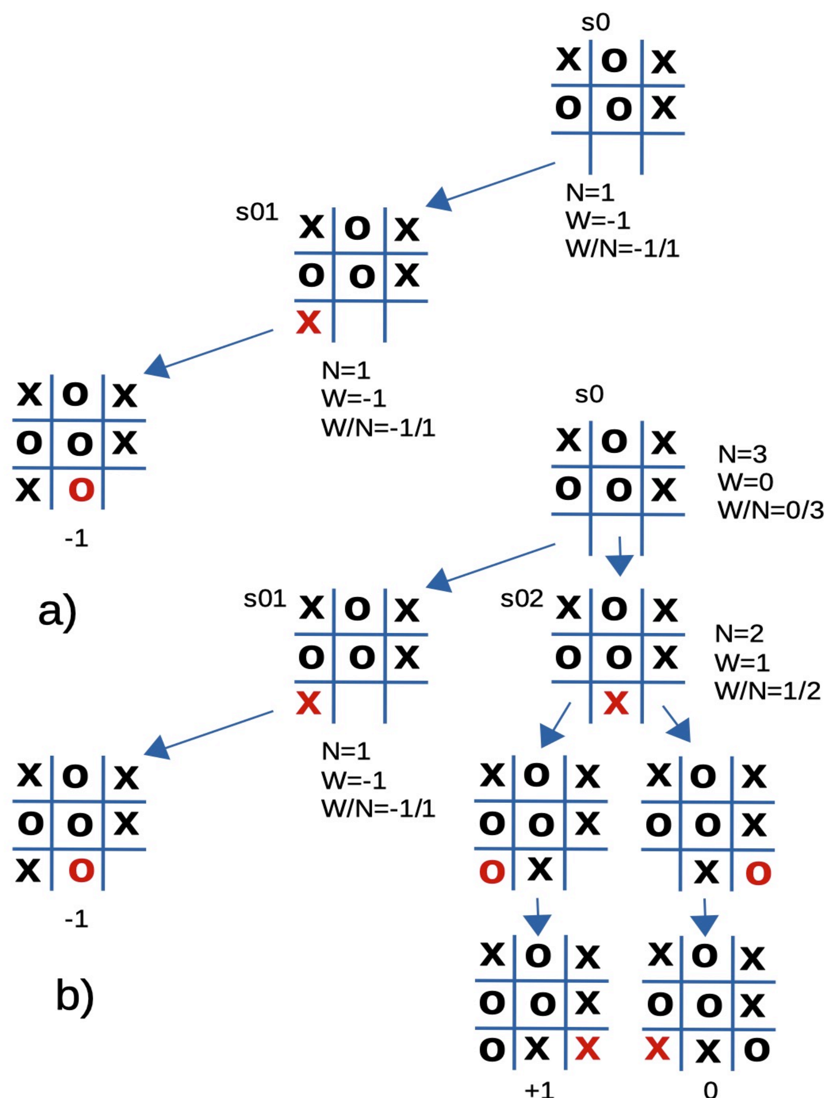
Ovo je jednostavni primjer kod kojeg smo napravili samo tri slučajne simulacije. Naravno, što je broj simulacija veći, to je i veća šansa da smo napravili dobru odluku. Kod Monte Carlo pretraživanja svakom simulacijom ažuriraju se sve težine na toj grani. Na primjer, da smo napravili sve moguće simulacije koje smo napravili kod MinMax pretraživanja za stanje *s0* prikazane na slici 3-23, čvorovi druge razine imali bi težine $u_{s01} = -1/2$, $u_{s02} = 1/2$ i $u_{s03} = 1/1$, pa bismo se kao i kod MinMax metode odlučili za *s03* koji vodi u najbržu pobjedu.

U prethodnom primjeru težinu čvora je određivao samo broj pobjeda (W_i) i broj simulacija (N_i) koje su startale od tog i-tog čvora

$$u_i = \frac{W_i}{N_i} \tag{3-10}$$

ali je uobičajeno da se kod Monte Carlo metode težina definira mjerom koja se zove ***gornja granica povjerenja*** (engl. *UPC – Upper Confidence Bound*):

$$u_i = \frac{W_i}{N_i} + c\sqrt{\frac{lnN_p}{N_i}} \tag{3-11}$$



Slika 3-24. Monte Carlo pretraživanje stabla. U prvom smo koraku (a) napravili jednu slučajnu simulaciju, koja je dala gubitak. Kada napravimo nove simulacije, također slučajnim izborom, trebamo unatrag ažurirati sve težine čvorova (vidi tekst).

gdje je N_i ukupan broj simulacija koje prolaze kroz i-ti čvor, N_p je broj simulacija roditeljskog čvora čvoru i, a parametar c (koji treba biti veći ili jednak nuli) određuje hoćemo li istraživati čvorove koje smo u simulacijama mali broj puta posjetili (veliki c) ili suprotno (mali c). Za naš prethodni primjer uz $c = 1$ i situaciju sa slike 3-24 pod b) težine čvorova *s01* i *s02* po ovom proračunu bi bile: $u_{s01} = -1/1 + (\ln 3/1)^{1/2} = 0{,}048$ i $u_{s02} = 1/2 + (\ln 3/2)^{1/2} = 1{,}241$, a u slučaju da su napravljene sve simulacije sa Slike 3-24: $u_{s01} = -1/2 + (\ln 5/2)^{1/2} = 0{,}397$, $u_{s02} = 1/2 + (\ln 5/2)^{1/2} = 1{,}397$ i $u_{s03} = 1/1 + + (\ln 5/1)^{1/2} = 2{,}269$.

Kod velikog broja simulacija Monte Carlo pretraživanje stabla konvergira prema MinMax pretraživanju, ali dosta sporo. Bez obzira na to ovaj postupak vodi prema puno manjem razvijenom prostoru rješenja zadatka od MinMax metode proširene alpha-beta obrezivanjem. Nadalje, Monte Carlo ne zahtijeva eksplicitnu heurističku funkciju. Težine se čvorova same računaju tijekom provođenja simulacija. Monte Carlo metoda je bila zaslužna što je ***AlphaGo*** pobijedio korejskog profesionalnog igrača ***Go***-a ***Lee Sedola*** rezultatom 4 : 1, iako se spominje i to da je četvrtu partiju izgubio zbog toga što postoje pojedinačne grane koje vode do gubitka, a slučajni izbor puteva ih ne može pronaći. Monte Carlo spada u postupke ***pojačanog učenja*** (engl. *Reinforcement Learning*) koje detaljno obrađujemo u Poglavlju 6.5.1.

3.6.3 Kombinacija optimalnog i heurističkog pretraživanje

I na kraju dolazimo do metoda pretraživanja kod kojih se kombinira optimalno i heurističko pretraživanje. U svakom pojedinom stanju s iz S kombiniramo do sada ostvarenu cijenu i predviđenu cijenu do cilja:

$$f(s) = c + h \tag{3-12}$$

gdje je c cijena do sada prijeđenog puta, a h procjena cijene do cilja.

Dva se algoritma najčešće koriste: ***algoritam optimalnog jednolikog troška proširen estimacijom*** i ***algoritam A****.

Pretraživanje optimalnim jednolikim troškom proširenim estimacijom

Pretraživanje optimalnim jednolikim troškom proširenim estimacijom (engl. *Estimate – Extended Optimal Uniform Cost Search*) razlikuje se od pretraživanja optimalnim jednolikim troškom samo po tome što se u svakom koraku novoformirani red sortira po vrijednosti kombinirane cijene pojedinog čvora koja uključuje do sada akumuliranu cijenu i estimaciju koliko još imamo do cilja (algoritam 3-9).

Algoritam 3-9. Pseudokod algoritma pretraživanja optimalnim jednolikim troškom proširenim estimacijom

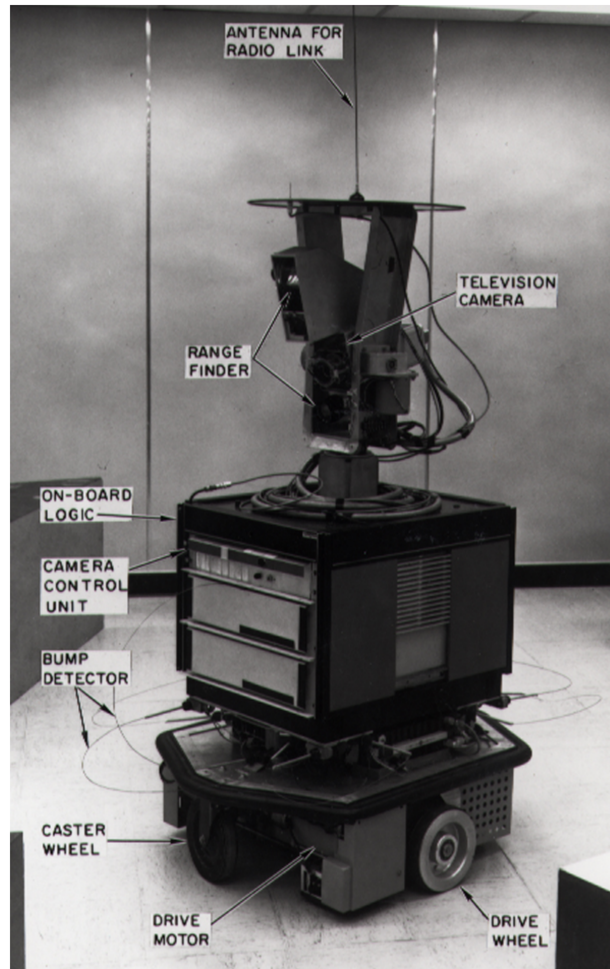
```
1. Pohrani početni čvor u niz.
2. DOK niz nije prazan i nije dosegnut cilj:
3. Skupi svu djecu prvog člana niza.
4. Odbaci djecu koja zatvaraju petlju.
5. Provjeri je li dosegnut cilj.
6. Izbriši prvi član reda, dodaj djecu te nakon toga POREDAJ NIZ PREMA VRIJEDNOSTI
   KUMULATIVNE FUNKCIJE f(s) = c(s) + h(s), tako da je na prvom mjestu najmanja
   vrijednost.
7. Ako smo došli do cilja, vrati USPJEH, ako nismo, vrati POGREŠKU.
```

A* pretraživanje

Algoritam pretraživanja A* *(čita se „Ei-Star")* je sigurno najznačajniji algoritam pretraživanja koji uz svoje različite izvedenice pronalazi primjenu kod brojnih problema koji se rješavaju metodama pretraživanja. Objavili su ga 1968. godine ***Peter Hart, Nils Nilsson*** i ***Bertram Raphael*** sa ***Stanford Research Institute*** kao dio istraživanja vezanog uz jednog od prvih mobilnih robota nazvanog ***Shakey*** prikazanog na Slici 3-25. A* algoritam je osmišljen kako bi se planiralo kretanje robota.

Algoritam je isti kao i pretraživanje optimalnim jednolikim troškom proširenim estimacijom, uz dodatak da se brišu redundantni putevi i to samo na temelju do sada akumulirane cijene. Pretpostavimo da se nakon širenja čvora nađemo u istom čvoru u kojem smo već bili na nekoj od prethodnih razina. Ako je akumulirana cijena tog čvora veća od akumulirane cijene tog istog čvora koji smo već otvorili, zaključak je da smo došli do istog čvora uz skuplju cijenu i da ga nema smisla dalje otvarati, pa se taj novi čvor isključuje iz daljnjeg pretraživanja.

Pseudokod algoritma daje algoritam 3-10:



Slika 3-25. Shakey, jedan od prvih mobilnih robota iz 1968. godine zbog kojeg je i osmišljen A algoritam*[29]

Algoritam 3-10. Pseudokod A algoritma*

```
1. Pohrani početni čvor u niz.
2. DOK niz nije prazan i prvi član niza nije dosegnut cilj:
3. Skupi svu djecu prvog člana niza.
4. Odbaci djecu koja zatvaraju petlju.
5. Provjeri je li dosegnut cilj.
6. Izbriši prvi član reda, dodaj djecu te nakon toga POREDAJ NIZ PREMA VRIJEDNOSTI
   KUMULATIVNE FUNKCIJE f(s) = c(s) + h(s), tako da je na prvom mjestu najmanja
   vrijednost.
7. Ako red sadrži čvor koji smo u prethodnoj liniji uključili, a njegova akumulirana
   cijena c(s) je veća od akumulirane cijene ovog novododanog čvora, taj novododani
   čvor izbriši.
8. Ako smo došli do cilja, vrati USPJEH, ako nismo, vrati POGREŠKU.
```

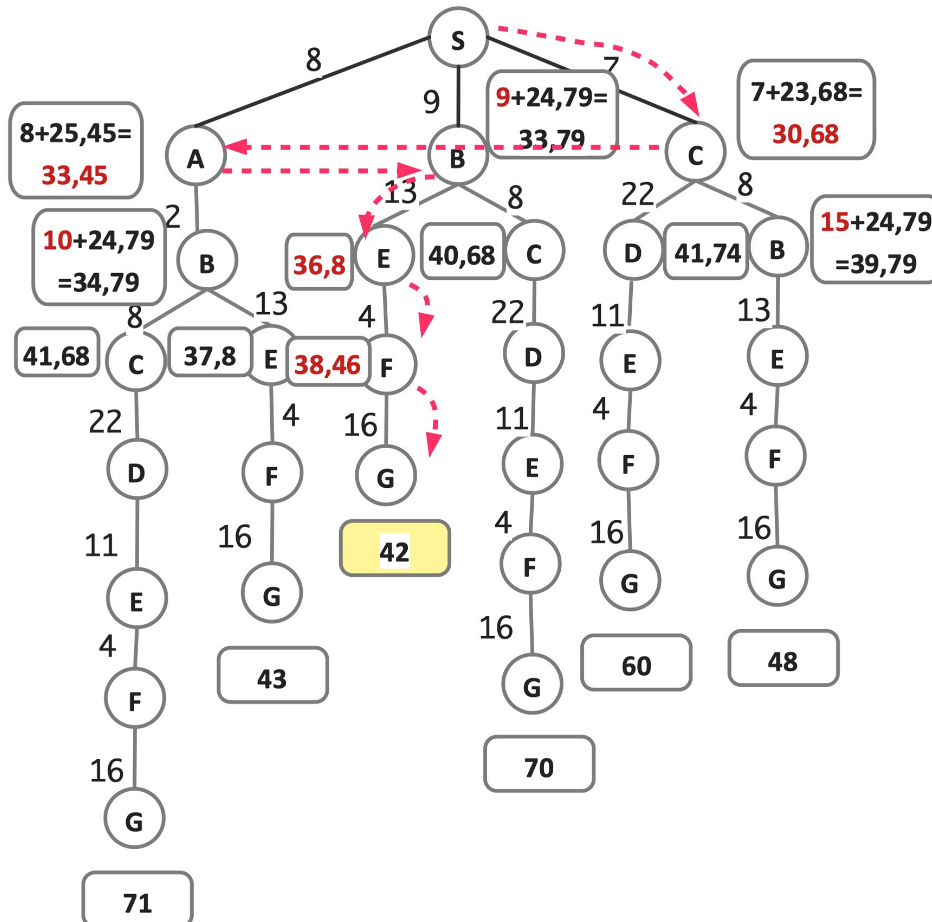
[29] Slika s https://commons.wikimedia.org/wiki/File:SRI_Shakey_with_callouts.jpg uz CC BY-SA 3.0 licencu.

U praktičnoj izvedbi kod A* algoritma obično se uvode:

closedSet – skup svih čvorova koji su već istraženi i prošireni, te

openSet – skup čvorova koji su stvoreni, ali još nisu prošireni. Na početku postupka pretraživanja *openSet* sadrži samo početni čvor *openSet* = { s_0 }.

Primjer primjene algoritma A* na traženju najkraćeg puta otoka Brača prikazuje slika 3-25.



Slika 3-25. Rješavanje zadatka najkraćeg puta na Braču metodom A pretraživanja*

Vremenska i prostorna složenost A* pretraživanja ovisi o heurističkoj funkciji, a u najgorem slučaju je eksponencijalna $O(b^d)$, gdje je *d* maksimalna dubina na kojoj se rješenje pronalazi. Problem je jedino ako rješenje ne postoji jer će ga A* algoritam i dalje stalno tražiti, pa je potrebno ugraditi vremensko ograničenje nakon kojeg se algoritam zaustavlja.

Varijacije A* pretraživanja

A* algoritam je potaknuo razvoj cijelog niza algoritama koji su uveli određene novosti ili prilagodili algoritam rješavanju specifičnih problema. Dijelimo ih u nekoliko različitih klasa[30]:

- inkrementalni algoritmi
- algoritmi koji vodu brigu o memoriji

[30] https://pdfs.semanticscholar.org/bc00/5547fd17d5e0880fa3d78a3cb6b99f41b719.pdf

- paralelni algoritmi
- algoritmi bilo kojeg vremena (engl. *„Anytime" algoritmi*) i
- algoritmi realnog vremena (engl. *„Real-Time" algoritmi*).

Inkrementalni algoritmi pretraživanja

Inkrementalni algoritmi pretraživanja (engl. *Incremental Search*) se dinamički mijenjaju i koriste informacije iz prethodnog pretraživanja za ubrzavanje trenutnog pretraživanja. Posebno su efikasni u dinamičkim okružjima koja se mijenjaju između dva koraka pretraživanja. Tri su različita pristupa inkrementalnim algoritmima pretraživanja:

- ***Algoritmi koji ponovo pokreću A**** u stanju kada trenutna pretraga odstupa od prethodne. Tipičan primjer je ***FSA**** (engl. *Fringe Saving A**)[31] iz 2007. godine.
- ***Algoritmi koji dinamički mijenjaju vrijednost heurističke funkcije*** *h(s)* (procjenu cijene do cilja). Kod ***A**** algoritma vrijednosti heurističke funkcije definiraju se na početku, a kod ove grupe algoritama mijenjaju se kako bi realnije procijenili udaljenosti do cilja. Primjer je ***GAA**** (engl. *Generalized Adaptive A**)[32] iz 2008. godine.
- ***Algoritmi koji dinamički mijenjaju cijenu postignutu od početka*** *c(s)* iz prethodnog pretraživanja ili, rečeno na drugačiji način, kod ovih se algoritama transformira stablo pretraživanja iz prethodnog pretraživanja u sadašnje pretraživanje. Primjeri su ***LPA**** *(Lifelong Planning A*)*[33] iz 2004. godine, ***D****, ponekad nazvan *Dinamički A* (Dynamic A*)*[34] iz 1995. godine, te ***D* Light***[35] iz 2002. godine. *D** algoritam i njegove izvedenice našli su praktičnu primjenu u brojnim stvarnim problemima kao što su navigacija autonomnih vozila na nepoznatom terenu, Mars Rover program itd.

Algoritmi koji vode brigu o memoriji

Algoritmi koji vodu brigu o memoriji (engl. *Memory-Concerned Search*) posebnu pažnju usmjeravaju prema količini memorije potrebne za pohranu čvorova u ***openSet*** i ***closedSet*** listama. Velika upotreba memorije jedna je od mana *A** algoritma, pa se različitim modifikacijama osnovnog algoritma nastojalo prevladati ovaj problem. I ovdje razlikujemo tri osnovna tipa algoritama:

- ***Memorijski efektivni algoritmi*** (engl. *Memory-Efficient Search*) ne drže u memoriji skupove *openSet* i *closedSet*, već otvaraju isti čvor onoliko puta koliko se pojavi tijekom pretraživanja. Problem je što dobitak na memoriji plaćaju dužim vremenom izvođenja. Primjeri su ***IDA**** *(Iterative-Deeping A*)*[36] iz 1985. godine i ***RBFS*** (engl. *Linear Space Best-First Search*)[37] iz 1993. godine.
- ***Memorijski ograničeni algoritmi*** (engl. *Memory-Bounded Search*) nastoje razriješiti problem dugog vremena izvođenja na način da ograničavaju veličinu korištene memorije kontrolirajući rast *openSet* skupa. Primjer je ***SMA**** (engl. *Simplified Memory-Bounded A**)[38]

---

[31] http://idm-lab.org/bib/abstracts/papers/ijcai07b.pdf

[32] https://www.cs.nmsu.edu/~wyeoh/docs/publications/aamas08-gaastar.pdf

[33] https://www.cs.cmu.edu/~maxim/files/aij04.pdf

[34] http://www.frc.ri.cmu.edu/~axs/doc/ijcai95.pdf

[35] https://pdfs.semanticscholar.org/9b5d/32d37b9809a295488e49c10919e312b5c357.pdf

[36] https://pdfs.semanticscholar.org/7eaf/535ca7f8d1e920e092483d11efb989982f19.pdf

[37] https://www.aaai.org/Papers/AAAI/1992/AAAI92-082.pdf

[38] https://cse.sc.edu/~mgv/csce580f09/gradPres/Russell_ecai92-sma.pdf

iz 1992. godine. Kod drugog pristupa spremaju se samo čvorovi nužni za traženje optimalnog rješenja. Primjer je ***PEA**** (engl. *Partial Expansion A**)[39] iz 2000. godine.

- ***Algoritmi s pomoćnom memorijom*** koriste sekundarnu memoriju za pohranu strukture algoritma. Sekundarna memorija (vanjski disk) daje puno više memorije za malu cijenu, ali čestim pisanjem i čitanjem sa sekundarne memorije znatno se usporava algoritam pretraživanja. Primjer je ***Frontier A****[40] iz 2004. godine.

Paralelni algoritmi

Paralelni algoritmi (engl. *Parallel Search*) prilagođeni su paralelnom izvođenju kako bi se pretraživanje vremenski ubrzalo. Većina ih je projektirana za arhitekture s distribuiranom memorijom kao na primjer ***PRA**** (engl. *Parallel Retracting A**)[41] algoritam iz 1991. godine. Sličan je i ***PLA**** (engl. *Parallel Local A**)[42] algoritam iz 1993. godine. Predloženi su i algoritmi koji koriste zajedničku memoriju.

Algoritmi bilo kojeg vremena

Osnovna značajka ***algoritama bilo kojeg vremena*** (engl. *„Anytime" Search*) je u tome da daju rješenje (ne nužno najbolje) u bilo kojem trenutku zaustavljanja algoritma nakon inicijalnog perioda kada se traži prvo rješenje. Kako vrijeme napreduje, algoritam daje sve bolje i bolje rješenje koje na kraju konvergira prema optimalnom rješenju. Primjeri su ***AWA**** *(Anytime Weighted A*)*[43] iz 2007. godine, ***ARA**** *(Anytime Repairing A*)*[44] iz 2004. godine i ***AD**** *(Anytime Dynamic A*)*[45] iz 2005. godine.

Algoritmi realnog vremena

Algoritmi realnog vremena (engl. *„Real-Time" Search*) su posljednja grupa modificiranih *A** algoritama kojima je osnovna značajka da mogu provoditi postupak pretraživanja uz prisustvo vremenskog ograničenja, ali pri tome ne garantiraju da će pronaći optimalno rješenje. Ponekad se ova skupina algoritama naziva i ***agentski usmjereno pretraživanje*** (engl. *Agent-Centered Search*). Koriste se u situacijama u kojima je vrijeme pretraživanja važnije od pronalaska najboljeg rješenja. Primjeri su ***LRTA**** *(Learning Real-Time A*)*[46] iz 1990. godine i ***RTAA**** *(Real-Time Adaptive A*)*[47] iz 2006. godine.

Rješavanje zadataka metodama pretraživanja imalo je, ima i imat će značajno mjesto u umjetnoj inteligenciji. Još uvijek postoje brojni zadaci koji se mogu riješiti jedino metodama pretraživanja, pa je dobro poznavanje temeljnih postupaka pretraživanja još uvijek vrlo važno kod savladavanja temeljnih znanja iz područja umjetne inteligencije.

....

[39] https://www.aaai.org/Papers/AAAI/2000/AAAI00-142.pdf

[40] https://www.aaai.org/Papers/AAAI/2004/AAAI04-103.pdf

[41] https://pdfs.semanticscholar.org/c412/a6b5c6ad577edd3c94096e99aa7216742ffe.pdf

[42] https://pdfs.semanticscholar.org/7e9c/e4feb25250713276eceb85166c10fe3bfb0e.pdf

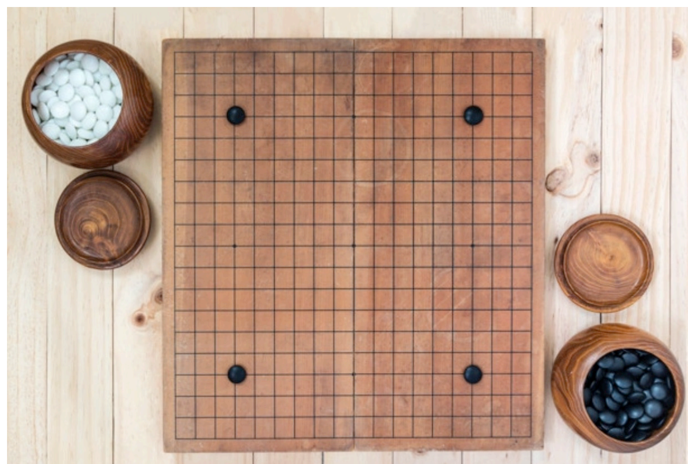
[43] https://www.aaai.org/Papers/JAIR/Vol28/JAIR-2808.pdf

[44] https://pdfs.semanticscholar.org/a931/74185b839bcad5e4f02d0ee65a449d1008ac.pdf

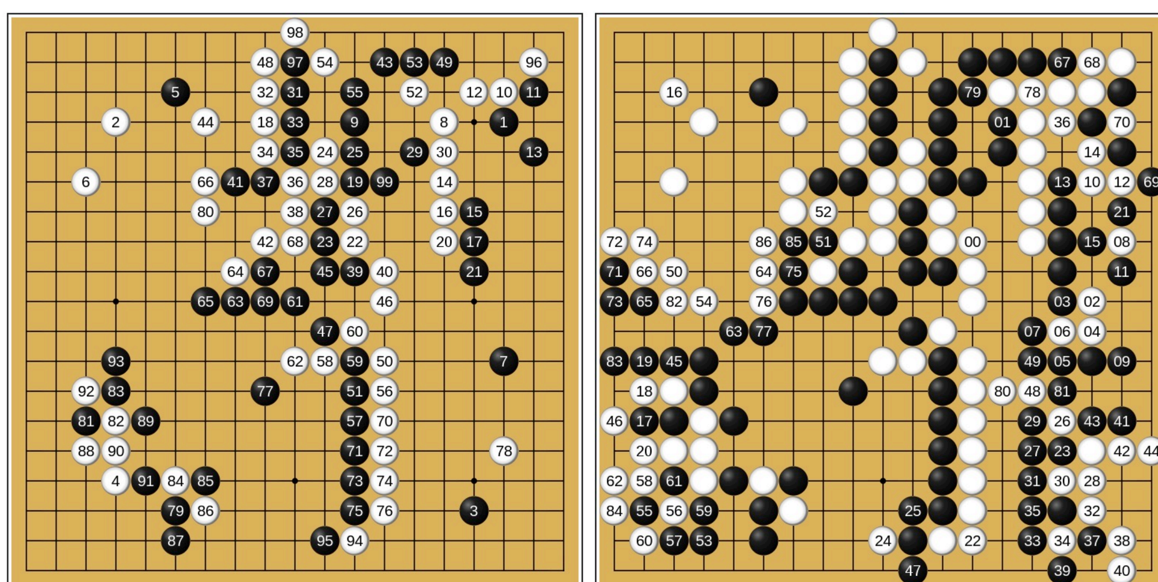
[45] http://www.cs.cmu.edu/~ggordon/likhachev-etal.anytime-dstar.pdf

[46] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.137.1955&rep=rep1&type=pdf

[47] http://www.cs.cmu.edu/~./motionplanning/papers/sbp_papers/integrated2/koenig_realtime_adaptive_astar_aamas06.pdf



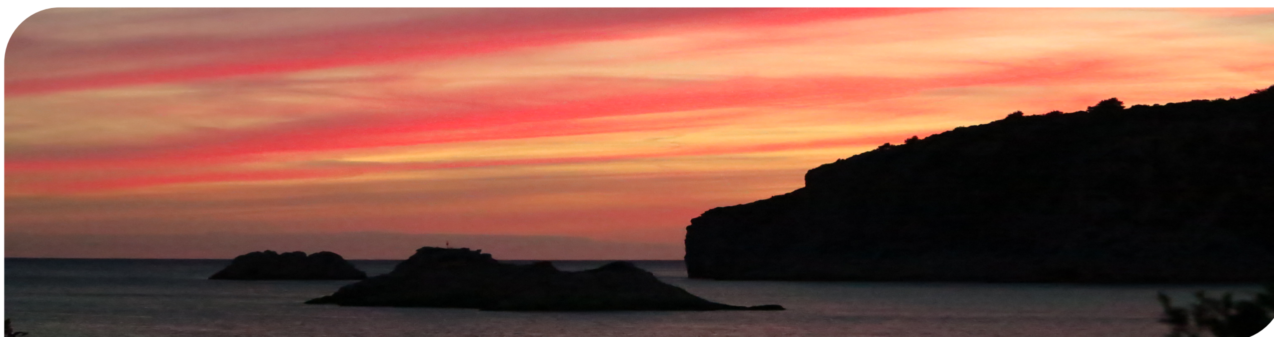
Dodatak: Go je kompleksna, apstraktna, strategijska igra dva igrača koji igraju crnim i bijelim figurama (žetonima) koji se na japanski zovi ***idi – kamen*** (碁石,棋子) na tabli ***goban***[48] (碁盤). Cilj igre je opkoliti protivničku figuru svojim figurama i na taj je način zarobiti. Igra je izmišljena u Kini prije više od 4000 godina, 2500 godina prije igre šah od koje je puno kompleksnija. Smatra se da igra šah ima 10^{120} mogućih kombinacija figura, a Go 10^{174}. Igra je posebno popularna na dalekom istoku. Od 9. – 15. ožujka 2016.g. u Južnoj Koreji odigrao se meč između svjetskog šampiona ***Lee Sedola*** koji je tu titulu ponio 18 puta i Googlovog programa ***AlphaGo*** poznatog i pod nazivom ***Google DeepMind Challenge Match***. Nagrada je bila 1 milion $, ali budući da je AlphaGo pobijedio s rezultatom 4:1 Google je nagradu donirao dobrotvornim organizacijama. Na slici ispod je položaj figura 1. partije nakon 99 poteza i za poteze od 100 – 186. AlphaGo se dosta razlikuje od programa koji su do tada korišteni kod igara dva igrača. Kod slavnog IBMovog računala ***Deep Blue*** koji je u igri šaha 1997.g. pobijedio ***Gary Kasparova*** koristilo se puno heuristike i MaxMin pretraživanje, dok se AlphaGo temeljio na umjetnim neuronskim mrežama i Monte Carlo pretraživanju stabla. Umjetne neuronske mreže korištene su kod predviđanja vjerojatnosti pobjede na temelju podataka o brojnim partijama igre Go, uključujući i partije koje je AlphaGo igrao sam protiv sebe. Postupkom Monte Carlo pretraživanja stabla računate su vjerojatnosti pobjede ili gubitka puno poteza u budućnosti. Više detalja o meču Sedola i AlphaGo možete pogledati u nagrađenom dokumentarnom filmu *„AlphaGo"*[49].



[48] Slika table je sa stranice https://www.freepik.com

[49] https://www.youtube.com/watch?v=WXuK6gekU1Y

4 PRIKAZIVANJE I POHRANA ZNANJA



Znanje je sigurno jedan od najvažnijih pojmova vezanih uz inteligentne tehnologije i nebiološku inteligenciju. Kod projektiranja inteligentnog sustava, bez obzira radi li se o sustavima temeljenim na (dobroj staroj) umjetnoj inteligenciji ili računskoj inteligenciji, potrebno je **znanje.** U biti, sustavi umjetne i računske inteligencije upravo se razlikuju po tome kako se znanje koristi. Prije odgovora na ovo pitanje najprije ćemo se upitati: "Što je znanje?"

Rješavanje kompleksnih problema i zadataka umjetne inteligencije zahtijeva ***veliku količinu znanja***, ali i mehanizme pomoću ***kojih manipuliramo*** tim znanjem. Sada dolazimo do problema znanja, prikazivanja znanja i pohrane znanja. Što je znanje? Kakvo znanje može biti? Kako znanje prikazati na način pogodan za pohranu na računalu? Sve su to pitanja na koja odgovaramo u ovom poglavlju.

4.1 Što je znanje?

Pojam ***znanja*** (engl. *Knowledge*) je jedan od onih filozofskih pojmova koji do dandanas nisu jednoznačno definirani. Klasična se definicija znanja pridjeljuje grčkom filozofu ***Platonu*** (427. – 347. g. pr. n. e.) koji je kazao da bi se neka izjava mogla smatrati znanjem samo ako je ta izjava ***„opravdano istinito vjerovanje*"**. Pod pojmom ***vjerovanje*** (engl. *Belief*) podrazumijeva se uvjerenost u istinitost bez prethodne provjere. Vjerovanje je subjektivna mentalna interpretacija do koje smo došli na temelju osjetilnih informacija, razmišljanja (zaključivanja), ili nam je komunikacijom to prenio netko drugi. ***Istina*** (engl. *Truth*) je jednako složen pojam kao i znanje koji također nema jedinstvene definicije. Zaustavimo se samo na tome da ona izražava neku vrstu poklapanja sa stvarnošću i realnošću. Treći je pojam ***opravdano*** (engl. *Justified*). Postoji cijela teorija koja se bavi upravo problemom opravdavanja vjerovanja, akcija, emocija, zahtjeva, teorija itd. Da bismo opravdali vjerovanje, treba nam dokaz.

Tipičan primjer vjerovanja i njegovog opravdanja nakon čega dolazimo do znanja je priča u kojoj partnerica (ili partner) vjeruje da joj druga strana nije vjerna. Tvrdnja *„Moj partner je nevjeran."* je vjerovanje sve dok se ne opravda dokazima. Dokaz može biti SMS poruka ili ostatak nepoznatog ruža na ovratniku partnerove košulje. Ovi dokazi opravdavaju vjerovanje, a ako su i istiniti i nisu rezultat slijeda slučajnosti, onda tvrdnja *„Moj partner je nevjeran."* postaje istinita, postaje znanje kojem možemo pridružiti vrijednost istinitosti, u ovom slučaju *„istina"*.

Platonova definicija znanja filozofima je zanimljiva i u današnje doba (Bezinović, 2006.). Cijela se znanost temelji na opravdavanju istinitosti. Svako znanstveno istraživanje počinje hipotezom. Hipoteza je tvrdnja u čiju istinitost vjerujemo, ali da bi ona postala znanje, moramo i opravdati tu istinitost znanstvenim dokazima. Tek nakon opravdavanja hipoteza postaje novo znanje. Ako se dokaže da hipoteza

nije istinita, ona je i tada znanje za koje sada znamo da je lažno. Istinitost tvrdnje može biti da je istinita ili da je lažna, ali ona je u oba slučaja postala znanje zato što joj možemo pridružiti stupanj istinitosti.

U nastavku ovog poglavlja (koje se prije svega bavi formalnim prikazom znanja pomoću matematičke logike) susretat ćemo se s vjerovanjem te opravdanjem i traženjem njegove istinitosti. Za sada spomenimo samo to da imamo vjerovanje P koje se može ili opravdati ili ne opravdati. Ako se može opravdati, tada uz pomoć njega možemo opravdati i novo vjerovanje Q koje je s njim povezano. Ako P ne možemo opravdati, tada iz njega ne možemo opravdati ni vjerovanje Q, ali isto tako ne možemo opravdati ni vjerovanje da Q nije istinito (*neQ*). Drugim riječima, o Q ne možemo ništa sigurno kazati ako se P ne može opravdati.

Uz pojam ***znanja*** usko su vezani i pojmovi ***podaci***, ***informacije*** i ***mudrost***, kao shvaćanje (razumijevanje) istinitosti znanja ilustrirano na slici 4-1.



Slika 4-1. Podaci, informacije, znanje i mudrost

Podaci (engl. *Data*) su simboli bez ikakvog značenja osim vlastitog postojanja. Mogu biti u različitim formama. Tipičan podatak je 23,4. On sam za sebe ništa ne znači. Radi se samo o slijedu tri broja između kojih je decimalni zarez. U računalnom svijetu podatke obrađuju tablični kalkulatori. Ono što nam je ovdje najzanimljivije jest pitanje možemo li podatku 23,4 pridružiti vrijednost istinitosti: *„Je li 23,4 istinito?"* Odgovor je *„Ne znam. Nemam dovoljno informacija da znam je li 23,4 istinito."* Povežemo li podatak 23,4 s nekim drugim podatkom iz drugog relacijski povezanog skupa, dobivamo informaciju.

Informacije (engl. *Information*) su obrađeni podaci koji su dobili značenje relacijskim vezama s drugim podacima. Ovako obrađeni podaci mogu biti korisni, iako to nije uvjet. U računalnom svijetu

informacije sadrže relacijsku bazu na temelju podataka posloženih u relacijske odnose. Ako je 23,4 spremljeno u tablici u kojoj pohranjujemo izmjerene vrijednosti temperature u stupnjevima C, onda znamo da se 23,4 odnosi na temperaturu od 23,4 ^{0}C i to je sada već informacija koju na određeni način možemo izvući iz te relacijske baze. Ponekad se kaže da informacije daju odgovore na pitanja *„tko", „što", „gdje"* i *„kada"*. Iz podataka dolazimo do informacija ako shvatimo odnose ili relacije između njih. Postavimo sada ponovo pitanje: *„Je li 23,4 ^{0}C istinito?"* Odgovor je ponovo: *„Ne znam. Nemam dovoljno informacija da znam je li 23,4 ^{0}C istinito. Trebaju mi informacije o tome kada i gdje je temperatura bila 23,4 ^{0}C."* Upravo to relacijsko povezivanje više informacija dovodi do tvrdnji kojima možemo pridružiti stupanj istinitosti, tj. dovodi do znanja.

Znanje (engl. *Knowledge*) je odgovarajući skup informacija kojima možemo pridružiti stupanj istinitosti. Tom skupu informacija osnovna je namjena da bude koristan. Znanje se stječe kroz proces učenja u kojem se informacije pretvaraju u znanje kako bi nam bilo od nekakve koristi, tj. kako bi nam poslužilo za nešto. Na primjeru prethodnog slučaja temperature od 23,4 ^{0}C znanje bi bilo povezivanje ovog podatka s podacima o trenutku mjesta i trenutku mjerenja, pa bi znanje bila tvrdnja *„Danas u 12 sati temperatura na meteorološkoj postaji Marjan Split bila je 23,4 ^{o}C."*. Ova je tvrdnja znanje zato što joj možemo pridružiti istinitost. Ako je temperatura stvarno danas u podne na Marjanu bila 23,4 ^{0}C, onda je tvrdnja istinita, a u protivnom je lažna. Znanje uvijek nečemu služi, ako ničemu drugom onda barem tome da se znamo primjereno odjenuti ako idemo prošetati popodne Marjanom. Ponekad se kaže da je znanje primjena podataka i informacija s ciljem davanja odgovora na određena pitanja. Iz informacija do znanja dolazimo ako shvatimo model, obrazac ili ***predložak*** (engl. *Pattern*). Na temelju njega možemo i predvidjeti što će se u budućnosti događati, a to kod samog poznavanja informacija nije moguće.

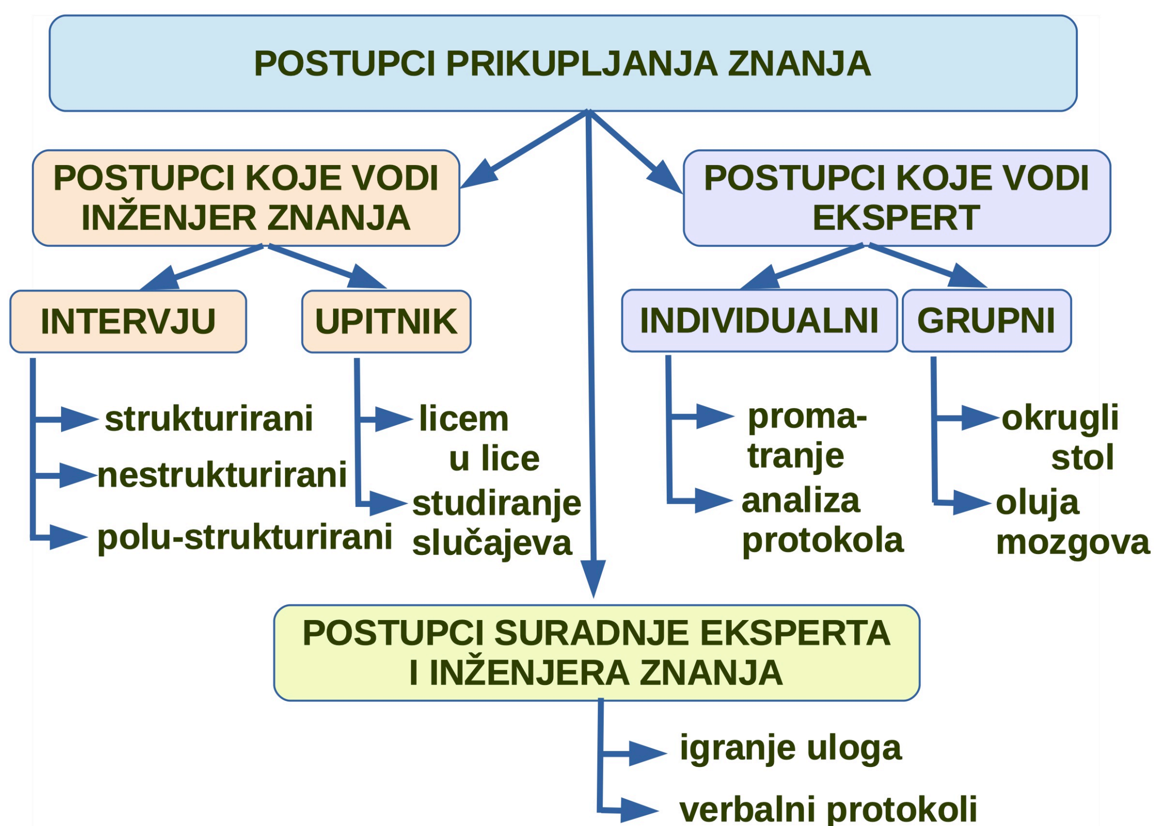
Sljedeći je korak shvaćanje toga zašto je tvrdnja istinita ili neistinita, a ovo nas shvaćanje vodi prema mudrosti. ***Mudrost*** (engl. *Wisdom*) uključuje shvaćanje temeljnih principa uključenih u znanje zbog kojih je znanje upravo takvo kakvo jest. Mudrost bi bila najveći stupanj spoznaje, koji bi trebao uključivati ne samo shvaćanje principa već i moral i etiku pa neki autori smatraju da mudrost može imati samo čovjek. Mi se ovdje ipak priklanjamo *„blažoj"* verziji pojma mudrost koja uključuje shvaćanje ne samo izjave, već i svega što stoji iza određene izjave. Mudrost bi bila kada bismo znali odgovoriti zašto je danas u podne na Marjanu temperatura bila (ili nije bila) 23,4 ^{0}C. Mudrost uključuje poznavanje modela i fizikalnih zakonitosti koji su rezultirali upravo ovakvom temperaturom danas u podne na Marjanu.

4.2 Prikupljanje, prikazivanje i pohrana znanja

Sustavi umjetne inteligencije spadaju u šire područje ***sustava temeljenih na znanju*** (engl. *KBS – Knowledge Based Systems*) koji se bave zadacima prikupljanja, prikazivanja, pohrane i primjene znanja. Ovi su postupci posebno važni za ekspertne sustave koji su jedno od područja primjene umjetne inteligencije. Oni se temelje na specifičnom znanju koje imaju posebni ljudi koje zovemo eksperti ili stručnjaci. Gradimo li na primjer specifični ekspertni sustav medicinske dijagnostike, temeljni je zadatak pronaći eksperta koji to znanje ima, pokušati od njega to znanje i izvući (prikupljanje znanja), te ga prebaciti u formalni oblik razumljiv računalu i pohraniti u specifičnu strukturu koju nazivamo ***baza znanja***.

Problemom ***prikupljanja znanja*** (engl. *Knowledge Elicitation*) bave se posebni stručnjaci koji se ponekad nazivaju i ***inženjeri znanja*** (engl. *Knowledge Engineers*), koji osim računalnog znanja imaju i specifična znanja psihologije i sociologije. Znanje se najčešće prikuplja u obliku rečenica prirodnog jezika zato što je to prirodni način komunikacije između ljudi. Različite su metode prikupljanja znanja, a svima je cilj doći do stvarnoga znanja eksperta. Dijelimo ih u tri grupe (slika 4-2):

- postupci prikupljanja znanja koje vodi inženjer znanja
- postupci koje vodi ekspert
- postupci u suradnji eksperta i inženjera znanja.



Slika 4-2. Podjela postupaka za prikupljanje ekspertnog znanja

Spomenut ćemo malo detaljnije neke od postupaka prikupljanja znanja:

1. ***Intervju*** *(*engl. *Interview)* – direktni razgovor između stručnjaka za prikupljanje znanja (inženjera znanja) i eksperta koji znanje ima. Ovo je možda najčešća metoda prikupljanja znanja kojoj je prije svega cilj doznati kako ekspert donosi pojedine odluke i provodi određene procedure. Sastoji se od pitanja i odgovora. Razlikujemo ***nestrukturirane, polustrukturirane i strukturirane intervjue***. Strukturirani intervjui vode se po unaprijed utvrđenom transkriptu (pripremljenim pitanjima), a nestrukturirani su u biti slobodni razgovor.

2. ***Analiza protokola*** *(*engl. *PA – Protocol Analysis)* – analiza postupka kako ekspert stvarno rješava zadatke. Stručnjak za prikupljanje znanja obično nije u direktnoj interakciji s ekspertom, već bilježi ili snima postupanja eksperta, te na temelju njih kasnije izvlači protokole i postupke. U postupcima analize protokola od eksperta se zahtijeva da i komentira ono što radi, a vezano s tim razlikuju se ***on-line*** i ***off-line*** postupci. Kod *on-line* postupaka ekspert komentira svoje postupke za vrijeme stvarnog rada. Varijanta *on-line* metode je da to ne radi ekspert sam, već drugi ekspert koji jako dobro zna što glavi ekspert radi. Ova se metoda obično naziva ***metoda sjene*** *(*engl. *Shadowing)*. Kod *off-line* postupaka ekspert naknadno komentira svoje postupke najčešće gledajući snimku onoga što je radio.

3. ***Proučavanje slučajeva*** (engl. *Case Study*) – temelji se na razgovoru s ekspertom o nekim specifičnim slučajevima, bilo standardnim, bilo složenijim. Kada se obrađuju slučajevi koji su bilo kritični tada se postupak zove ***metoda kritičnih odluka*** (engl. *CDM – Critical Decision Method*). Ovi postupci često uključuju i intervju i analizu protokola, ali u kontekstu specifičnih slučajeva koji su se ekspertu pojavili u prošlosti, pa je za njih trebao donijeti odluku ili provesti određeni postupak.

Osim ova tri postupka spomenimo da ih postoji još, npr. ***sortiranje koncepata*** (engl. *Concept Sorting*), ***ponavljajuće mreže*** (engl. *Repetitory Grid*), ***stepeničaste mreže*** (engl. *Laddered Grids*), ***simulacije*** (engl. *Simulations*), ***postupak 20 pitanja*** (engl. *20 Questions*), ali oni prelaze okvire ovog udžbenika.

Nakon prikupljanja znanja ono je obično zapisano riječima i rečenicama prirodnog jezika, u više-manje slobodnom tekstu. Sljedeći je zadatak znanje prebaciti u formalni zapis kako bismo ga relativno jednostavno računalno zapisali. Ovaj se postupak naziva ***prikazivanje*** ili ***kodiranje znanja*** (engl. *Knowledge Coding*), a najčešće se provodi korištenjem matematičke logike, od klasične propozicijske i predikatne, do različitih nestandardnih logika od kojih je posebno neizrazita logika našla posebno mjesto u praksi.

Treći je korak formalno kodirano znanje ***pohraniti*** (engl. *Knowledge Storing*) u strukture koje zovemo ***baze znanja*** (engl. *Knowledge Bases*). Osnovna im je uloga omogućiti što lakši i jednostavniji dolazak do pojedinih dijelova znanja tijekom provođenja postupka ***zaključivanja*** (engl. *Inference*) i ***rasuđivanja*** (engl. *Reasoning*) temeljenog na znanju. Ovoj smo problematici, zbog njene izuzetne važnosti, posvetili cijelo Poglavlje 5.

O načinima pohrane znanja više se detalja može pronaći u nastavku teksta, nakon što se osvrnemo na osnovne postupke prikazivanja ili kodiranja znanja matematičkom logikom.

4.3 Prikazivanje (kodiranje) znanja matematičkom logikom

Važan pojam u svim sustavima prikaza znanja su ***činjenice*** – stvari koje predstavljamo. U logičkim sustavima činjenice su tvrdnje čiju istinitost poznajemo. Kako bismo mogli s jedne strane činjenice pripremiti za predstavljanje u formalnim prikazima znanja, a s druge strane i koristiti dobivene rezultate, nužno je uspostaviti vezu između formalnog prikaza i vanjskog svijeta. Najprirodnija takva veza su riječi i rečenice prirodnog jezika, pa bez obzira kako mi interno činjenice prikazujemo u bazama znanja, čovjeku je najrazumljivije predstavljanje toga znanja prirodnim jezikom. Na sličan način radi i naš mozak. Iako još uvijek potpuno pouzdano ne znamo kako mozak sprema znanje, znamo da čovjek najlakše svoje pohranjeno znanje iskazuje prirodnim, govornim jezikom. Pogledajmo primjer jezičnog iskazivanja i logičkog prikazivanja nekih tvrdnji:

prirodni jezik: *„Rex je pas"*

*logičko prik*azivanje (propozicijska logika): *„Rex = pas"*

prirodni jezik: *„Rex ima rep."*

*logičko p*rikazivanje (predikatna logika): *„ima_rep(Rex)"*

Logički formalizam izuzetno je važan zbog činjenice što se ***novo znanje može generirati iz starog znanja*** koristeći postupak ***matematičkog zaključivanja***. Logika barata istinitostima tvrdnji i daje precizna pravila kako se može odrediti istinitost složene tvrdnje sastavljene od pojedinačnih tvrdnji poznate istinitosti povezanih logičkim operatorima. Pomoću logike u stanju smo ustanoviti istinitost neke tvrdnje čiju istinitost ne poznajemo, ako dokažemo da je ta tvrdnja izvedena iz činjenica koje su istinite i poznate. Logike dijelimo na standardne logike, koje se oslanjaju na klasične logike *(**propozicijsku** ili **predikatnu logiku**)*, i nestandardne logike, od kojih je možda najpoznatija ***neizrazita*** (engl. ***fuzzy***) ***logika*** koja je u posljednje vrijeme nezaobilazna kod mnogih primjena inteligentnih tehnologija. Standardna je logika dvovaljana, tj. ima samo dva stupnja istinitosti, pa tvrdnja može biti ili istinita ili lažna (neistinita). Trećeg nema. Neke od nestandardnih logika su viševaljane, tj. mogu imati više stupnjeva istinitosti. Granični slučaj je neizrazita (engl. *fuzzy*) logika koja je beskonačno valjana. Više detalja o ovim razlikama u nastavku ovog poglavlja. Za sada se zadržavamo na standardnim logikama.

Grčkog filozofa i prirodnjaka ***Aristotela*** (384. – 322. pr. Kr.) smatraju prvim priznatim logičarom. On je razvio veći dio teorije koja je danas poznata pod pojmom ***klasična logika, silogistička logika*** ili ***Aristotelova logika***. U svojoj osnovi silogistička se logika bavi pronalaženjem istinitosti (ili neistinitosti) na temelju filozofskih argumenata. Danas se još uvijek često upotrebljava zato što su na njoj izgrađene gotovo sve legalne argumentacije. Pogledajmo primjer:

Pretpostavke (poznate činjenice)

Ivan je muškarac.

Svi muškarci nekad su bili dječaci.

Zaključak (izvedena činjenica)

Ivan je nekad bio dječak.

Pretvoreno u formu silogističkog zaključivanja gornji se zadatak može simbolički pisati na sljedeći način:

Ivan = muškarac

svi muškarci → dječaci

Ivan = dječak

Silogistička logika na neki je način samo formalni zapis zdravog razuma, a koliko je dobra ili nije, to je potpuno drugo pitanje. Kako se zasnivala na prirodnom jeziku, a on je po svojoj naravi neprecizan i često dvosmislen, ponekad je ovakvo logičko zaključivanje dovodilo do konfuzije. To je možda prednost u pravnim sustavima, ali je u tehnici nedostatak, pa se javila potreba za definiranjem logike koja bi bila preciznija.

Takva je logika nazvana ***simbolička logika***, a otac joj je bio ***Gottfried Wilhelm Leibniz*** (1646. – 1717.). Međutim, njegovom smrću rad na simboličkoj logici zaustavlja se na više od 100 godina. Oživljava ga tek ***George Boole*** (1815. – 1864.), te se u spomen na njega simbolička logika često i naziva ***Boolova logika***, ali se često spominje i kao ***matematička logika***. Simbolička logika barata simbolima (koji su apstrakcija koncepata), činjenicama te vezama između simbola, odnosno činjenica, a na način da se definiraju operatori baratanja simbolima. U nastavku se bavimo simboličkom interpretacijom propozicijske logike i predikatne logike.



Slika 4-3. Aristotel, otac klasične logike, Gottfried Wilhelm Leibniz, otac simboličke logike i George Bool, otac matematičke logike[50]

U osnovi računalnog programiranja nalazi se upravo logika. Programski su jezici po mnogim svojim značajkama samo implementacija specijalnih logičkih formi. Na primjer, u jeziku C logiku predstavljaju generalne procedure iskazane IF naredbama, dok je jezik Prolog primjer direktne programske implementacije predikatne logike.

Simbolička logika smatra da se znanje može prikazati ***simbolima***, a zaključivanje se odvija ***manipulacijom simbola*** čime je omogućeno apstraktno rasuđivanje. Formalni logički sustav sastoji se

[50] Sve su slike s Wikipedia Commons https://commons.wikimedia.org s licencom Public Domain.

od sintakse, semantike i teorije dokaza. ***Sintaksa*** određuje po kojim se pravilima slažu složeniji logički izrazi i formiraju u tzv. ***dobro formulirane formule*** (engl. *wff – well-formed formulas*). ***Semantika*** se bavi značenjem logičkih izraza i određivanjem njihove istinitosti, dok ***teorija dokaza*** definira postupke kojima se provodi zaključivanje i rasuđivanje. Sintaksa i semantika omogućavaju prikazivanje ili kodiranje znanja iz stvarnoga svijeta koji nas okružuje, a teorija dokaza omogućava izvođenje novog znanja iz pohranjenog znanja postupcima zaključivanja i rasuđivanja.

Alfred North Whitehead i ***Bertrand Russell*** su u periodu 1910. – 1913. godine napisali knjigu u tri volumena s naslovom ***Principia Mathematica*** kao pokušaj da se skupom aksioma i pravila zaključivanja simboličke logike dokažu sve matematičke istine. Slika 4-4 prikazuje primjer logičkog dokaza iz knjige da je 1 + 1 = 2. Knjiga je bila vrlo važan doprinos matematici i filozofiji.

$*54{\cdot}43.\ \vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$

Dem.

$\vdash . *54{\cdot}26 . \supset \vdash :. \alpha = \iota{`}x . \beta = \iota{`}y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$

$[*51{\cdot}231] \qquad \equiv . \iota{`}x \cap \iota{`}y = \Lambda .$

$[*13{\cdot}12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash . (1) . *11{\cdot}11{\cdot}35 . \supset$

$\vdash :. (\exists x, y) . \alpha = \iota{`}x . \beta = \iota{`}y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$

$\vdash . (2) . *11{\cdot}54 . *52{\cdot}1 . \supset \vdash .$ Prop

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

Slika 4-4. Logički dokaz da je 1 + 1 = 2 iz knjige „Principia mathematica" autora Alfreda Whiteheada i Bertranda Russella[51]

Jedini je problem bio u tome što ova knjiga nikada nije mogla biti dovršena zato što ni jedan formalni aksiomatski sustav nije sposoban modelirati aritmetiku što je 1931. godine dokazao matematičar i filozof ***Kurt Gödel*** poznatim ***teoremima nepotpunosti*** (engl. *Gödel's Incompleteness Theorems*). Za umjetnu inteligenciju to i nije toliko važno, ali poznavanje osnovnih principa matematičke logike izuzetno je važno, pa se u nastavku kratko opisuju osnovni principi standardne i nekoliko nestandardnih logika s posebnim naglaskom na njihovu primjenu u sustavima pohrane znanja umjetne inteligencije, a zaključivanjem i rasuđivanjem bavimo se u Poglavlju 5.

4.3.1 Propozicijska logika

Propozicijska logika ili ***logika sudova*** je najjednostavnija logika kod koje se činjenice prikazuju logičkim propozicijama. Na primjer:

prirodni jezik: *Pada kiša.* logički prikaz: *kiša*

prirodni jezik: *Sunčano je.* logički prikaz: *sunčano*

prirodni jezik: *Kada pada kiša, nije sunčano.* logički prikaz: *kiša* → ¬ *sunčano*

gdje je ¬ logički operator negacije, a → logički operator implikacije.

Propozicijska se logika bavi određivanjem istinitosti ili neistinitosti različitih propozicija, a propozicija je ispravno postavljena tvrdnja koja može biti istinita ili lažna. Propozicije se međusobno povezuju operatorima u ***dobro formulirane formule***. Operatora ima ukupno 16, od kojih 15 povezuju dvije propozicije *(binarni operatori)*, a samo se operator negacija odnosi na samo jednu varijablu *(unarni*

[51] Slika iz https://commons.wikimedia.org/wiki/File:Principia_Mathematica_54-43.png uz licencu Public Domain.

operator). Deset binarnih operatora imaju i posebno ime, a ostalih pet su komutacije postojećih operatora. Oznake pojedinih operatora, naziv operatora i jezični primjer su u nastavku.

$\bullet$ tautologija – $P\dot{}Q$ - „P je Q” – „*Pada kiša ili ne pada kiša.*”

$\vee$ disjunkcija -$P \vee Q$ - „P ili Q” – „*Pada kiša ili je zemlja mokra.*”

$\rightarrow$ implikacija – $P \rightarrow Q$ - „ako je P onda je Q” – „*Ako pada kiša onda je zemlja mokra.”*

$\leftrightarrow$ ekvivalencija – $P \leftrightarrow Q$ - „P je onda i samo onda ako je Q” – „*Zemlja je mokra onda i samo onda ako pada kiša.*”

$\wedge$ konjunkcija – $P \wedge Q$ – „P i Q” – „*Pada kiša i zemlja je mokra.”*

$|$ Schefferova operacija – $P \mid Q$ – „nije istodobno i P i Q” – „*Nije istodobno i da pada kiša i da je zemlja mokra.*”

ex ekskluzivna disjunkcija – $P\ ex\ Q$ – „P ili Q, ali nije oboje” – „*Pada kiša ili je zemlja mokra, ali ne oboje.*”

$\downarrow$ Piersova operacija (ili Lukasiewiczova operacija) – $P \downarrow Q$ – „niti je P niti je Q” – „*Niti pada kiša niti je zemlja mokra.*”

^{o} kontradikcija – $P^{o}Q$ – „P nije Q” - „*Pada kiša i ne pada kiša.*”

$\rightsquigarrow$ nema službenog imena pa ćemo je nazvati valovita implikacija – $P \rightsquigarrow Q$ – „ako nije P onda je Q” – „*Ako ne pada kiša, onda je zemlja mokra.*”

$\neg$ negacija – $\neg P$ – „nije P” - „*Ne pada kiša.*”

Istinitost dvije tvrdnje povezane operatorima računa se pomoću tablica istinitosti prikazuje tablica 4-1. Broj 1 znači „*istina*”, a 0 znači „*nije istina*”. Napomenimo da se često ***istina*** označava i simbolima $\top$, *t* ili *true*, a ***laž (neistina)*** simbolima $\bot$, *f* ili *false*.

Tablica 4-1. Tablica istinitosti tvrdnji povezanih različitim logičkim operatorima klasične Aristotelove logike

P	Q	$P\dot{}Q$	$P \vee Q$	$Q \rightarrow P$	P	$P \rightarrow Q$	Q	$P \leftrightarrow Q$	$P \wedge Q$
1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	0	0	0	0
0	1	1	1	0	0	1	1	0	0
0	0	1	0	1	0	1	0	1	0
P	Q	$P \mid Q$	$P\ ex\ Q$	$\neg Q$	$Q \rightsquigarrow P$	$\neg P$	$P \rightsquigarrow Q$	$P \downarrow Q$	$P\overset{o}{}Q$
1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0
0	1	1	1	0	0	1	1	0	0
0	0	1	0	1	0	1	0	1	0

Od svih ovih operacija za umjetnu inteligenciju najznačajnije su operacije negacije, konjunkcije, disjunkcije, implikacije i ekvivalencije.

Složene propozicije konstruiramo iz više propozicija povezanih operatorima. Na primjer, složenu tvrdnju:

((A i B) ili (C i neD))

logički zapisujemo:

$$((A \wedge B) \vee (C \wedge \neg D))$$

Ovako zapisan logički izraz je dobro formulirana formula. Dobro formulirana formula je i logički izraz

$$((P \rightarrow Q) \rightarrow (\neg Q \rightarrow \neg P))$$

Ako je na primjer propozicija P tvrdnja „*Objekt je stol.*", a propozicija Q tvrdnja „*Objekt je namještaj.*" ovu formulu možemo jezično interpretirati:

„*Ako je objekt stol, onda je on namještaj, a ako objekt nije namještaj nije ni stol.*"

S druge strane logički izraz

$((P \rightarrow Q) \rightarrow (PP)Q))$

nije dobro formulirana formula i ne možemo ga ni na koji način suvislo jezično interpretirati. Propozicije u drugom dijelu implikacije nisu povezane važećim logičkim operatorima, a izraz čak ima i jednu zatvorenu zagradu viška.

Zagrade određuju redoslijed operacija, a istinitost se traži primjenom operatora od lijeva na desno. Ako zagrade nisu označene postoji dogovor o redoslijedu izvođenja operacija: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$. Prvo se izvodi operacija negacije, zatim konjunkcija, disjunkcija, implikacija i na kraju ekvivalencija. Postupak utvrđivanja istinitosti dobro formulirane formule naziva se ***interpretacija formule*** i ona određuje značenje (semantiku) složene logičke propozicije. Pretpostavimo da su u logičkom izrazu $((A \wedge B) \vee (C \wedge \neg D))$ tvrdnje *A, B i D* istinite, a da tvrdnja *C* nije istinita. Tablice istinitosti za pojedine logičke operatore daju:

A	B	C	D	¬ D	(A∧B)	(C∧¬ D)	(A∧B) ∨ (C∧¬D)
1	1	0	1	0	1	0	1

što znači da je složena propozicija *((A i B) ili (C i neD))* istinita.

Kod formalnog zapisa interpretacije formule možemo koristiti i funkcijske oznake *t(.)* i ***t(.)***[52] koje se međusobno u interpretaciji razlikuju. *t(.)* je funkcija koja pojedinim propozicijama pridružuje vrijednosti istinitosti, a ***t(.)*** je funkcija pomoću koje interpretiramo istinitost složenih propozicija (dobro formuliranih formula). Obje funkcije poprimaju vrijednosti na dvočlanom skupu $\{0,1\}$, gdje kao i prije *0* znači laž, a *1* znači istina. Za prethodni slučaj imamo:

$t(A)=1,\ t(B)=1,\ t(C)=0,\ t(D)=1$

$\mathbf{t}((A \wedge B) \vee (C \wedge \neg D)) = (t(A) \wedge t(B)) \vee (t(C) \wedge \neg t(D)) = (1 \wedge 1) \vee (0 \wedge \neg 1) = (1 \wedge 1) \vee (0 \wedge 0) = 1 \vee 0 = 1$

S obzirom na poznati redoslijed operacija $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ formulu $((A \wedge B) \vee (C \wedge \neg D))$ smo mogli pisati i bez zagrada kao $(A \wedge B \vee C \wedge \neg D)$.

Ako se dvije složene propozicije sastoje od istih osnovnih propozicija i imaju istu tablicu istinitosti za njih kažemo da su ***ekvivalencije*** i povezujemo ih znakom ≡. Pri tome treba uočiti razliku između ekvivalencije ≡ i operatora ekvivalencije ↔. Ekvivalencija ≡ samo kaže da se lijeva i desna strana sastoje od istih propozicija i imaju istu tablicu istinitosti, a operator ekvivalencije ↔ je operator definiran svojom tablicom istinitosti. Na primjer, propozicije $(P \wedge Q)$ i $(Q \wedge P)$ su ekvivalencije pa možemo pisati $(P \wedge Q) \equiv (Q \wedge P)$, a $(P \wedge Q) \leftrightarrow (Q \wedge P)$ označava novu propoziciju *R* koja ima svoju vrijednost istinitost, ovisno o tome kakve su istinitosti propozicija *P* i *Q*. Najvažnije ekvivalencije imaju i svoje ime, a koriste se kod pojednostavljivanja složenih propozicija:

KOMUTATIVNOST	$P \wedge Q \equiv Q \wedge P$
	$P \vee Q \equiv Q \vee P$
ASOCIJATIVNOST	$(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$
	$(P \vee Q) \vee R \equiv P \vee (Q \vee R)$
DE MORGANOVI TEOREMI	$\neg (P \vee Q) \equiv \neg Q \wedge \neg P$
	$\neg (P \wedge Q) \equiv \neg Q \vee \neg P$
DISTRIBUTIVNOST	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$

[52] Prva je funkcija oznaka zapisana običnim slovima, a druga pojačanim slovima (bold).

	$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
APSORPCIJA	$P \vee (P \wedge Q) \equiv P$
	$P \wedge (P \vee Q) \equiv P$
	$P \vee (\neg P \wedge Q) \equiv (P \vee Q)$
	$P \wedge (\neg P \vee Q) \equiv (P \wedge Q)$
ZAKON KONTRADIKCIJE	$P \wedge \neg P \equiv 0$
TAUTOLOGIJA	$P \vee \neg P \equiv 1$
DVOSTRUKA NEGACIJA	$\neg\neg P \equiv P$
IDEMPOTENCIJA	$P \wedge P \equiv P$
	$P \vee P \equiv P$
KONTRAPOZICIJA	$(P \rightarrow Q) \equiv (\neg Q \rightarrow \neg P)$
ELIMINACIJA UVJETA	$(P \rightarrow Q) \equiv (\neg P \vee Q)$
ELIMINACIJA DVOSTRUKOG	$(P \leftrightarrow Q) \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$

Svaku od ovih ekvivalencija možemo i jezično interpretirati. Pogledajmo primjer kontrapozicije. Rečenica: *„Ako je predmet stol, onda je on namještaj."* je implikacija. Njena kontrapozicija koja je u svim kombinacijama istinitosti ekvivalentna s osnovnom implikacijom je rečenica *„Ako predmet nije namještaj, onda on nije stol."*. Ili pogledajmo primjer eliminacije uvjeta. Implikacija *„Ako ti ne odeš, onda ću otići ja."* je za sve vrijednosti istinitosti identična (ekvivalentna) tvrdnji *„Otiđi ili idem ja."*. Uz dvostruku negaciju vezana je i poznata šala vezana uz dalmatinski govor. Upitate li osobu u Dalmaciji hoće li na kavu s vama, odgovor je često *„A ne neću."* što u biti znači *„Hoću."*.

Važan dio propozicijske logike su ***pravila zaključivanja*** na temelju kojih provodimo postupak ***logičkog rasuđivanja*** (engl. *Logical Reasoning*) i to posebno ***automatiziranog logičkog rasuđivanja*** (engl. *Automated Reasoning*) o čemu govorimo u Poglavlju 5. Ovdje spomenimo samo to da se u postupku logičkog zaključivanja korištenjem tablica istinitosti dokazuje da zaključak slijedi iz određenih pretpostavki. Na taj se način može automatizirati postupak izvođenja istinitosti tvrdnji čiju istinitost ne poznajemo na temelju poznavanja istinitosti drugih tvrdnji koje su logički povezane s tvrdnjama nepoznate istinitosti.

Iako je propozicijska logika osnova digitalne elektronike, sklopovskog ustroja računala i brojnih računalnih jezika, ona nije baš najpogodnija za prikazivanje ljudskog znanja zato što nema mogućnosti prikazivanja odnosa između objekata, pa se korištenjem propozicijske logike ne može upotrijebiti nikakva klasifikacija.

Pogledajmo primjer dviju propozicija:

prirodni jezik: *Ivan je čovjek.* → logički prikaz: *Ivan = čovjek*

prirodni jezik: *Luka je čovjek.* → logički prikaz: *Luka = čovjek*

Na desnoj strani imamo dvije nezavisne propozicije koje logički nemaju ništa zajedničko, bar u okviru propozicijske logike. Međutim, mi znamo da one i te kako imaju zajedničkih svojstava, pa bi bilo prirodnije tvrdnje *„Ivan je čovjek."* i *„Luka je čovjek."* napisati u obliku svojstvenih izjava *čovjek(Ivan)* i *čovjek(Luka)*, a to je način izražavanja tvrdnje u okviru ***predikatne logike***.

4.3.2 Predikatna logika

Propozicijska logika ne dozvoljava formiranje generalnih izjava tipa *„Marko jede sve što mu je drago."*. U duhu propozicijske logike za sva jela koja Marko voli trebamo napisati posebne propozicije, na primjer *„Marko voli čokoladu."*, *„Marko voli kekse."*, *„Marko voli sir."*, *„Marko ne voli pršut."* itd. Predikatna logika to omogućava. Umjesto da se bavi propozicijama koje se ne mogu raščlaniti,

predikatna logika uvodi pojam ***predikata*** koji je u biti funkcija koja vraća vrijednost istinitosti (0 ili 1) u ovisnosti o svojim argumentima.

Na primjer, u prethodnom primjeru uvodimo predikat *voli(.,.)* koji ima dva argumenta. Prvi se argument odnosi na osobu, a drugi na jelo koje ta osoba voli ili ne voli. Očito je da će funkcija *voli(marko,čokoladu)* vratiti vrijednost istinitosti 1, zato što Marko voli čokoladu, a *voli(marko,pršut)* vratiti vrijednost istinitosti 0, zato što Marko ne voli pršut.

Za drugi primjer uzmimo predikat *je_tvrd(.)* koji ima samo jedan argument. Ubacivanjem argumenta kamen, funkcija *je_tvrd(kamen)* vraća vrijednost istinitosti istina (1), a ubacivanjem argumenta pamuk *je_tvrd(pamuk)* vraća vrijednost istinitosti laž (0). Formalno to pišemo:

je_tvrd(kamen)=1

je_tvrd(pamuk)=0

a jezična interpretacija je u biti ista kao u propozicijskoj logici:

„Kamen je tvrd."

„Pamuk nije tvrd."

Osnovna razlika između predikatne i propozicijske logike je ***odvajanje atributa od objekata*** koji posjeduju te atribute. Drugim riječima, u predikatnoj logici može se definirati funkcija kojom određujemo tvrdoću bilo kojeg objekta, dok se u propozicijskoj logici za svaki pojedini slučaj mora definirati posebna tvrdnja.

Predikatna logika ima dosta zamršenu povijest. ***Charles Sanders Pierce*** (1839. – 1914.), američki filozof, logičar i matematičar, je 1885. godine prvi put predikatnu logiku spomenuo kao poseban logički sustav, naslanjajući se na rad Georga Boola, iako je i prije njega 1879. godine njemački filozof, logičar i matematičar ***Gottlob Frege*** (1848. – 1925.) u logiku uveo pojmove relacija i kvantifikatora, što propozicijska logika nije imala. Ovi su radovi uglavnom zaboravljeni dok ih 1917. godina nije ponovo popularizirao u svojim predavanjima ***David Hilbert*** (1862. – 1943.).

Osnovni pojmovi predikatne logike su:

- ***Domena*** je skup elemenata nad kojima se izvode zaključivanje, na primjer skup ljudi.
- ***Konstante*** su elementi domene. U prethodnom primjeru to su pojedini ljudi koji imaju svoja imena. Mi ćemo ih označavati malim slovima, pa je domena koju čine ljudi s imenima Marko, Ivan i Marija skup {*marko, ivan, marija*}, a *marko, ivan* i *marija* su konstante te domene.
- ***Varijable*** su simboli koje obično označavamo slovima *X, Y, W* itd., a koji mogu poprimiti bilo koju vrijednost iz domene.
- ***Predikat*** je funkcija koja preslikava jedan ili više elemenata domene u jednu od vrijednosti istinitosti (istina, 1, T, *t*, *true* ili laž, 0, ⊥, *f*, *false*). Predikat govori o nekom svojstvu elemenata domene ili o međusobnim relacijama elemenata domene. Na primjer, *je_tvrd(kamen)* vraća stupanj istinitosti tvrdnje da konstanta *kamen* zadovoljava svojstvo *je_tvrd*. Ovakav predikat sa samo jednom konstantom zove se „***unarni predikat***". Predikati mogu imati i veći broj argumenata, npr. predikat *voli(X,Y)* gdje se *X* odnosi na osobu, a *Y* na jelo, ili *jednak(X,Y)* koji vraća 1 samo ako su *X* i *Y* jednaki, ili *veći(X,Y)* koji vraća vrijednost istinitosti 1 samo ako je X > Y. Predikat s dvije konstante zove se „***binarni predikat***".

Ovome možemo dodati i pojam ***funkcije*** koja preslikava jednu ili više konstanti natrag u domenu. Na primjer, *ZBROJ(X,Y)* čija je domena skup cijelih brojeva i koja vraća zbroj konstanti *X* i *Y*, na primjer *ZBROJ(2,3)* = *5*, pa će na primjer predikat *veći(ZBROJ(2,3),4)* vratiti vrijednost istinitosti 1, zato što je *ZBROJ(2,3)* veći od *4*.

Konstante, varijable, funkcije i predikati čine 4 disjunktna skupa predikatne logike. Konstante, varijable i funkcije još se nazivaju i ***izrazi*** (engl. *Terms)*, a predikati zajedno sa svojim argumentima ***atomi*** (engl. *Atoms)*. Atomi povezani logičkim operatorima, na primjer *(parni(3)* ∨ *veći(3,4))*, čine ***logičku formulu***.

Najveći doprinosi predikatne logike su ***upotreba varijabli*** i uvođenje ***kvantifikatora***. Pomoću varijabli generaliziramo predikat, npr.

voće(X) → ¬ povrće(X)

što čitamo: „*Ako je X voće, tada X nije povrće.*"

Predikatna logika razlikuje dva kvantifikatora:

∀ - univerzalni kvantifikator koji se čita „*za svaki*" i

∃ - egzistencijalni kvantifikator koji se čita „*postoji*" ili „*egzistira*".

Na primjer, za tvrdnju „*Sve mačke su životinje.*" pišemo:

∀X. (mačke(X) → životinje(X))

ili tvrdnju: „*Postoji osoba koja se zove Ivan.*" pišemo:

∃X. (osoba(X) ∧ ime(X, Ivan))

I kvantifikatori imaju svoje ekvivalencije, što znači da je istinitost tvrdnji na lijevoj i desnoj strani ista:

$\neg\forall X.\ P(X) \equiv \exists X.\neg P(X)$

$\neg\exists X.\ P(X) \equiv \forall X.\neg P(X)$

$\forall X.\ \forall Y.P(X,Y) \equiv \forall Y.\ \forall X.P(X,Y)$

$\exists X.\ \exists Y.P(X,Y) \equiv \exists Y.\ \exists X.P(X,Y)$

$\forall X.P(X) \wedge \forall X.Q(X) \equiv \forall X.(P(X) \wedge Q(X))$

$\exists X.P(X) \vee \exists X.Q(X) \equiv \exists X.(P(X) \vee Q(X))$

Kvantifikatori se mogu odnositi samo na varijable, kao u prethodnim primjerima, a mogu se odnositi i na varijable i na predikate. Pod pojmom ***predikatne logike*** obično se podrazumijeva prvi slučaj kada se kvantifikatori odnose samo na varijable. Ovakav se tip logike naziva još i ***logika prvog reda*** ili skraćeno ***FOL*** (čita se *ef-ou-el*) prema engleskim riječima ***First-Order Logic***. Sinonimi su i ***predikatna logika prvog reda*** ili kraće ***FOPL – Fist-Order Predicate Logic*** i ***niža predikatna logika***. Ako se kvantifikatori mogu odnositi i na predikate, tada se govori o ***logici drugog reda*** (***SOL – Second-Order Logic***). U okviru ovog teksta zadržavamo se samo na predikatnoj logici prvog reda.

Jedan od značajnih jezika umjetne inteligencije ***Prolog*** u potpunosti se zasniva na predikatnoj logici. Predikatnu logiku koristimo i za formalni prikaz znanja i za formalno prikazivanje pravila na osnovi kojih dolazimo do zaključka. Sjetimo se zadatka dva vrča. U njemu smo logičkim formulama definirali uvjete prelaska iz jednog stanja unutar prostora rješenja zadatka u drugo stanje. Važno je napomenuti da se u predikatnoj logici ***vrijeme ne prikazuje***. Na primjer, predikat *pisati(krleža,zastave)* znači i „*Krleža je napisao zastave.*" i „*Krleža piše zastave.*" i „*Krleža će napisati zastave.*".

Predikatna logika dozvoljava da se ista tvrdnja prikaže na različite načine. Na primjer, tvrdnju „*Kuća je žute boje.*" možemo prikazati kao:

žuta(kuća)

boja(kuća, žuta)

vrijednost(boja, kuća, žuta)

Predikatna logika u potpunosti se naslanja na propozicijsku logiku, pa više-manje sve što smo kazali za propozicijsku logiku vrijedi i za predikatnu logiku, uz to da sada imamo i kvantifikatore.

Ako u formuli neku od varijabli povežemo s kvantifikatorom, tada kažemo da je ta ***varijabla vezana***. U protivnom je ***varijabla slobodna***. Na primjer, u formuli *∀X. (P(X,Y))* varijabla *X* je vezana, a varijabla *Y* slobodna. Kvantifikatori imaju prednost u odnosu na logičke operatore, a redoslijed izvođenja logičkih operatora je isti kao kod propozicijske logike. Određivanje istinitosti formule i ovdje se zove ***interpretacija formule***. Tablice istinitosti logičkih

operatera su iste kao kod propozicijske logike, pa ostaje jedino pitanje interpretacija formula koje sadrže varijable i kvantifikatore.

Značenje (interpretacija) kvantifikatora ∀ definira se na način da je formula u kojoj imamo vezanu varijablu ovim kvantifikatorom istinita za sve konstante iz domene te varijable. Neka je predikat *voli(X,čokoladu)*, a varijabla *X* poprima vrijednosti na domeni {*marko, ivan, marija*}. Formula ∀*X.voli(X,čokoladu)* znači da su istinite sve tri tvrdnje *voli(marko,čokoladu)*, *voli(ivan,čokoladu)* i *voli(marija,čokoladu)*.

Značenje (interpretacija) kvantifikatora definira se na sličan način, samo što je dovoljno da je jedan od prije navedenih predikata istinit. Ako u prethodnom primjeru samo Marko voli čokoladu, a Ivan i Marija je ne vole, formula ∀*X. voli(X,čokoladu)* nije istinita, ali je formula *X. voli(X,čokoladu)* istinita zato što postoji predikat *voli(marko,čokoladu)* koji je istinit.

Predikatna logika ima i svoja specifična ***pravila zaključivanja*** o kojima ćemo više govoriti u dijelu o zaključivanju. U ovom poglavlju zanima nas prije svega kako formulom predikatne logike formalno prikazati znanje. Neke smo primjere već vidjeli pa ih sada pokušajmo sistematizirati.

Prikazivanje jednostavnih činjenica (tvrdnji)

Tvrdnje oblika „*Marko voli čokoladu.*" ili „*Ivan voli Mariju.*" su jednostavne za prikazivanje u duhu predikatne logike. Uobičajeno je koristiti glagol kao predikat, a imenice kao argumente – *voli(marko,čokoladu)* ili *voli(ivan,mariju)*. Na sličan način prikazujemo i tvrdnje koje opisuju osobinu, na primjer boju – „*Kuća je žuta.*", s tim da se sada svojstvo (atribut) uzme za predikat, a imenica za argument *žuta(kuća)*. Složene tvrdnje formiramo koristeći logičke operatore. Na primjer, „*Kuća je žuta i velika.*" u duhu predikatne logike iskazujemo formulom *(žuta(kuća) ∧ velika(kuća))*. Ovdje je važno napomenuti da se logički operatori ne mogu koristiti kod povezivanja argumenata, već samo kod povezivanja predikata. Formula *(voli(marko,sir) ∧ voli(marko,pršut))* je ispravna, a formula *(voli(marko,sir ∧ pršut))* nije.

Prikazivanje uzročno – posljedičnih veza

Uzročno -posljedične veze su posebno važne. Cijeli niz ekspertnih sustava temelji se na prikazu uzročno-posljedičnih odnosa između varijabli. „*Ako je Marko gladan, onda će on jesti kruh.*" u duhu predikatne logike prikazujemo uvođenjem predikata *gladan(X)* i *jesti(X,Y)* koji su povezani operacijom implikacije:

gladan(marko) → jesti(marko,kruh)

Kvantifikatore koristimo kako bismo izraz poopćili. Neka je domena varijable *X* skup ljudi. *„Ako su ljudi gladni, onda jedu kruh."* iskazujemo formulom

∀X.(gladan(X) → jesti(X,kruh))

Naravno da se mogu koristiti različite kombinacije logičnih operatora. Pogledajmo nekoliko složenijih primjera i njihov prikaz u duhu predikatne logike:

„Svatko voli nekoga."

∀X.(osoba(X) → ∃Y. (osoba(Y) ∧ voli(X,Y))

„Netko je voljen od svih."

∃X. ∀Y.(voli(X,Y))

„Postoji stol koji nema 4 noge."

∃X.(stol(X) ∧ ¬ broj_noga(X,4))

„Ne postoji voće koje je papreno."

¬ ∃X.(voće(X) ∧ papreno(X))

Kako bismo na primjer mogli prikazati pojam djeda u duhu predikatne logike? Trebamo definirati tri predikata s dva argumenta: *djed(X,Y), otac(X,Y)* i *majka(X,Y)* koji znače „*X je djed od Y*", odnosno „*X je otac od Y*" i „*X je majka od Y*". Važno je znati na koga se pojedini argument odnosi. Obično se na prvom mjestu pojavljuje argument koji se u jezičnom iskazu javlja prvi. Pojam djeda sada možemo definirati logičkom formulom:

$$\forall X.\ \forall Y.\ \forall Z\ .\ ((otac(X,Z) \wedge otac(Z,Y)) \wedge (otac(X,Z) \wedge majka(Z,Y)) \rightarrow djed(X,Y))$$

Na kraju napomenimo još jednom da se programski jezik ***Prolog*** u potpunosti oslanja na predikatnu logiku i ima ugrađene sve mehanizme predikatnog računa pomoću kojih se provodi automatsko logičko zaključivanje. Pri tome je formalizam prikaza malo drugačiji pa se primjerice logička implikacija

$$((otac(X,Z) \wedge otac(Z,Y)) \rightarrow djed(X,Y))$$

u Prologu označava

```
djed(X,Y):-
    otac(X,Z),
    otac(Z,Y).
```

Na prvom je mjestu obično posljedični dio implikacije, a iza znaka :- uzročni. U standardnom logičkom prikazu implikacija se interpretira:

„Ako je X otac od Z i Z otac od Y, onda je X djed od Y."

a u Prolog obliku:

„X je djed od Y ako je X otac od Z i Z otac od Y."

Predikatna logika je dobar matematički aparat za prikaz znanja, ali postoji cijeli niz tvrdnji koje se ne mogu prikazati predikatnom logikom, a često su baš takve informacije važne u sustavu umjetne inteligencije. Upravo se zbog toga pokušalo znanje prikazati i nekim drugim logikama. Neke od logika koje ćemo spomenuti u nastavku koristimo kod prikaza tvrdnji tipa:

„Temperatura je danas vrlo visoka." – Kako prikazati tvrdnju koja iskazuje ***relativni stupanj*** u ovom slučaju temperature.

„Ljudi plave kose obično imaju plave oči." – Kako prikazati ***iznos sigurnosti***.

„Bolje je pomaknuti više figura nego protivnik." – Kako prikazati ovu ***heurističku informaciju***.

„Znam da Ivan misli da će Marko pobijediti, ali ja mislim da će on izgubiti." – Kako prikazati ***više različitih vjerovanja odjednom***.

Ljudi često barataju vjerovanjima ili ***uvjerenjima*** (engl. *Beliefs*). Svako od njih je podržano odgovarajućim dokazima, ali nije rijetkost da je skup vjerovanja nepotpun ili ponekad čak nekonzistentan. Kako koristiti nepotpune, nejasne, nesigurne informacije s kojima predikatna logika ne može izaći na kraj? Upravo zbog toga rješenja su tražena u drugim, tzv. ***nestandardnim logikama*** (standardne logike su propozicijska i predikatna logika).

4.3.3 Nestandardne logike

Nestandardne logike možemo podijeliti u dvije osnovne grupe (Turner, 1984.):

1. Logike ***koje proširuju standardne logike*** i nisu s njima u kontradikciji. Kod njih vrijede svi aksiomi standardnih logika plus neki dodatni. Tipičan primjer su ***modalne logike***, na primjer ***osnovna modalna logika*** koja uvodi nove operatore: L (nužno je) i M (moguće je), zatim ***temporalne logike*** kojima se dodaje pojam vremena, neke ***trovaljane logike*** itd.

2. Logike koje su ***u suprotnosti sa standardnim logikama***, kod kojih se pojedini aksiomi suprotstavljaju aksiomima standardnih logika. Tipičan primjer su ***viševaljane logike***, kao na primjer ***Kleenova, Łukasiewiczeva, Bochvarova, neizrazita*** (engl. ***fuzzy***) ***logika*** itd.

U ovom se poglavlju u najkraćim crtama osvrćemo na neke od njih.

Modalna logika

Modalne logike uvode tzv. modalitete, odnosno barataju konceptima kao što su mogućnost, nužnost, eventualnost, možda, mora, može itd. Prva od njih je osnovna ***modalna logika*** koja uvodi samo koncepte nužnosti i mogućnosti. Nastala je kao proširenje klasične predikatne logike uvođenjem operatora *L* (ili ◊) (koji se interpretira ***nužno je*** (engl. *Necessarily*)) i operatora *M* (ili ☐) (koji se interpretira kao ***moguće je*** (engl. *Possibly*)). Ako je neka tvrdnja *P* važeća u okviru predikatne logike, tada je i tvrdnja *LP* i *MP* važeća u okviru modalne logike, a interpretira se na sljedeći način:

P - tvrdnja A je točna

LP - nužno je da je tvrdnja A točna

MP - moguće je da je tvrdnja A točna

Operatori su međusobno ovisni, pa jednog možemo izračunati iz drugog uz dodatnu primjenu operatora negacije.

$LP \equiv \neg M \neg P$

$MP \equiv \neg L \neg P$

Kaže se da je neka tvrdnja ***moguća*** ako bi mogla biti istinita, bez obzira što je ili nije istinita u ovom trenutku. Isto tako tvrdnja je ***nužna ako nije moguće da bi bila lažna***. Na sličan način možemo kazati da je tvrdnja ***moguća ako nije nužno da je ona lažna***.

Modalna logika u umjetnoj se inteligenciji ponekad koristi kod modeliranja zaključivanja o znanju, iako se često ističe da je previše ograničena, pa se modaliteti pokušavaju uvesti na druge načine.

Temporalna logika

Temporalna logika također spada u modalne logike i nastaje kao proširenje standardnih logika. Osnovni razlog njenog uvođenja je taj što predikatna logika nije kadra baratati s informacijama kod kojih je važan pojam vremena. U temporalnoj logici ista tvrdnja može imati različitu vrijednost istinitosti u različitim vremenima – tvrdnja može npr. biti istinita neki trenutak u prošlosti, a u ovom trenutku neistinita i opet postati istinita u budućnosti. Postoji više različitih temporalnih operatora, a dijelimo ih na kvantifikatore (*A*, *E*) i operatore (*X, G, F, U*). Kvantifikatori i operatori uvijek idu u paru (na primjer *AX* ili *EF*), a značenje im je sljedeće:

Kvantifikatori

A – za sve *(**A**ll)*

E – postoji bar jedan *(**E**xists)*

Operatori

N – prvo sljedeće stanje - prvo sljedeće vrijeme *(**N**ext)*

G – sva stanja - sva vremena *(**G**lobally)*

F – postoji stanje - neko vrijeme u *budućnosti (**F**inally)*

U – od ovog stanja do nekog budućeg stanja - od ovog vremena do nekog budućeg vremena *(**U**ntil)*

Pogledajmo primjer. Neka su *p* i *q* propozicije koje su točne, na primjer:

p = *„Ja volim čokoladu."*

q = *„Vani je vruće."*

Kombinacija temporalnih kvantifikatora i operatora daje:

AG.p – *„Voljet ću čokoladu, u svim vremenima."*

EF.p – *„Možda ću voljeti čokoladu jedan put u budućnosti."*

A(pUq) – „Od sada pa dok ne postane vani vruće, voljet ću čokoladu svaki dan."

Temporalna se logika u umjetnoj inteligenciji koristi za modeliranje domena koje uključuju pojam vremena, ali se isto tako koristi i u računarskim znanostima za predviđanje ponašanja računalnog programa. Osim temporalne logike u okviru umjetne inteligencije predloženi su i drugi postupci pomoću kojih se može uvesti koncept vremena, na primjer ***John McCarthy*** je predložio tzv. ***situacijski račun*** (engl. *Situation Calculus*).

Viševaljane logike

Kod ***viševaljanih*** ili ***viševalentnih logika*** (engl. *Many-valued, Multi-valued, Multiple-valued ili Multivalent Logics*) uvode se dodatni stupnjevi istinitosti. U svim dosadašnjim logikama tvrdnja je mogla biti samo istinita ili lažna, vrijednosni skup stupnjeva istinitosti imao je samo dva člana {0,1}, gdje 0 znači laž, a 1 istina. Ovo pravilo, koje u klasičnoj logici nazivamo ***princip isključenja trećega*** ili ***zakon isključenja srednjega*** (engl. *Principle of Excluded Third* ili *Law of Excluded Middle*) formulirao je i sam Aristotel preko ***principa nekontradikcije*** (engl. *PNC - Principle of Non-contradiction*). Jedna od njegovih tvrdnji iz Metafizike glasi: *„Najsigurnije od svih osnovnih načela jest da proturječne tvrdnje nisu istinite istodobno."*. Sud može bili ili istinit, ili lažan, trećega nema. Logički se to iskazuje ***zakonom kontradikcije*** $(P \wedge \neg P \equiv 0)$ ili ***tautologijom*** $(P \vee \neg P \equiv 1)$. Stoička filozofska škola se zalagala za strogu primjenu ovoga zakona, dok su već i epikurejci dozvoljavali mogućnost da ni jedan od dvaju sudova koji se međusobno negiraju ne bude istinit (ne vrijedi tautologija). Problem se aktualizira početkom dvadesetog stoljeća kada se predlažu različiti sustavi viševaljane logike kod kojih se vrijednosni skup istinitosti proširuje. To je moguće napraviti na dva načina:

- dodavanjem novih vrijednosti koje nisu u kontradikciji s aksiomima klasične logike, pa se u tom slučaju radi samo o proširenoj klasičnoj logici koja razrješava njene pojedine nedostatke ili
- dodavanjem novih vrijednosti istinitosti koje su u suprotnosti s aksiomima klasične logike, posebno sa principom isključenja trećeg.

Postoji cijeli niz ***trovaljanih logika*** nastalih proširenjem klasične logike dodavanjem treće vrijednost istinitosti. Tipičan primjer je ***Kleenova logika*** koju je predložio logičar ***Stephen Kleen*** (1909. – 1994.), a koja uvodi treću vrijednost istinitosti koja se naziva ***neodlučeno*** (engl. *Undecided*) i označava simbolom *?* ili slovom *u*. Neodlučeno znači da se ne može odlučiti o istinitosti tvrdnje. Uvođenjem vrijednosti *?* potrebno je proširiti tablice istinitosti za sve operatere, pa se primjerice tablice istinitosti negacije, konjunkcije, disjunkcije i implikacije definiraju tablicom 4-2. Prisjetimo se da se implikacija može zamijeniti s negacijom i disjunkcijom korištenjem eliminacije uvjeta: $(P \rightarrow Q) \equiv (\neg P \vee Q)$

Tablica 4-2. Tablica istinitosti logičkih operatora u Kleenovoj trovaljanoj logici

$\neg P$	
1	0
0	1
?	?

$P \wedge Q$			
P \ Q	1	0	?
1	1	0	?
0	0	0	0
?	?	0	?

$Q \vee P$			
P \ Q	1	0	?
1	1	1	1
0	1	0	?
?	1	?	?

$P \rightarrow Q$			
P \ Q	1	0	?
1	1	0	?
0	1	1	1
?	1	?	?

Drugi primjer je ***Łukasiewiczeva trovaljana logika*** koju je predložio poznati poljski matematičar ***Jan Łukasiewicz*** (1878. – 1956.). Ona uvodi treću vrijednost istinitosti nazvanu ***neodređeno*** (engl. *Indeterminate*) i označenu slovom *i*. Pojam ***neodlučno*** razlikuje se od pojma ***neodređeno***. Neodlučeno znači da postoji praznina u istinitosti, da se ***ne možemo odlučiti*** je li istinito ili ne, a moglo bi biti ili jedno ili drugo, dok neodređeno znači da se istinitost ***ne može odrediti***, odnosno problem nije u tome da ne znamo vrijednost istinitosti samo zbog toga što nemamo dovoljno informacija, već nismo uopće u mogućnosti proračunati vrijednost istinitosti. Motivacija Łukasijewicza je bila ukloniti

nemogućnost Aristotelove logike da bilo što kaže o istinitosti nečega što će se dogoditi u budućnosti. Tablice istinitosti za operacije negacije, konjunkcije i disjunkcije su iste kao i kod Kleenove logike, ali se razlikuje implikacija. Definira je tablica 4-3. Razlika je u tome da je implikacija istinita ako neodređeno implicira neodređeno.

Tablica 4-3. Tablica istinitosti implikacije u Lukasiewiczevoj trovaljanoj logici

P→ Q			
P \ Q	1	0	*i*
1	1	0	*i*
0	1	1	1
i	1	*i*	1

Treći primjer trovaljane logike koja je nastala kao proširenje klasične logike je ***Bochvarova logika*** koju je predložio ***Dmitrii Antolevich Bochvar*** (1903. – 1990.) inspiriran semantičkim paradoksom. Uzmimo za primjer tvrdnju „*Ova tvrdnja je lažna.*". Ako je istinita, tada mora biti lažna, a ako je lažna, tada mora biti istinita. Da bi riješio ovaj paradoks, Bochvar uvodi treću vrijednost koja nije ni istinita ni lažna, već jednostavno ***paradoksalna*** ili ***besmislena*** (engl. *Meaningless*), a označava se slovom *m*. Operacije negacije, konjunkcije, disjunkcije i implikacije u slučaju Bochvarove logike definiraju se tablicom 4-4.

Tablica 4-4. Tablica istinitosti logičkih operatora u Bochvarovoj trovaljanoj logici

¬ *P*	
1	0
0	1
m	*m*

P ∧ *Q*			
P \ Q	1	0	*m*
1	1	0	*m*
0	0	0	*m*
m	*m*	*m*	*m*

Q ∨ *P*			
P \ Q	1	0	*m*
1	1	1	*m*
0	1	0	*m*
m	*m*	*m*	*m*

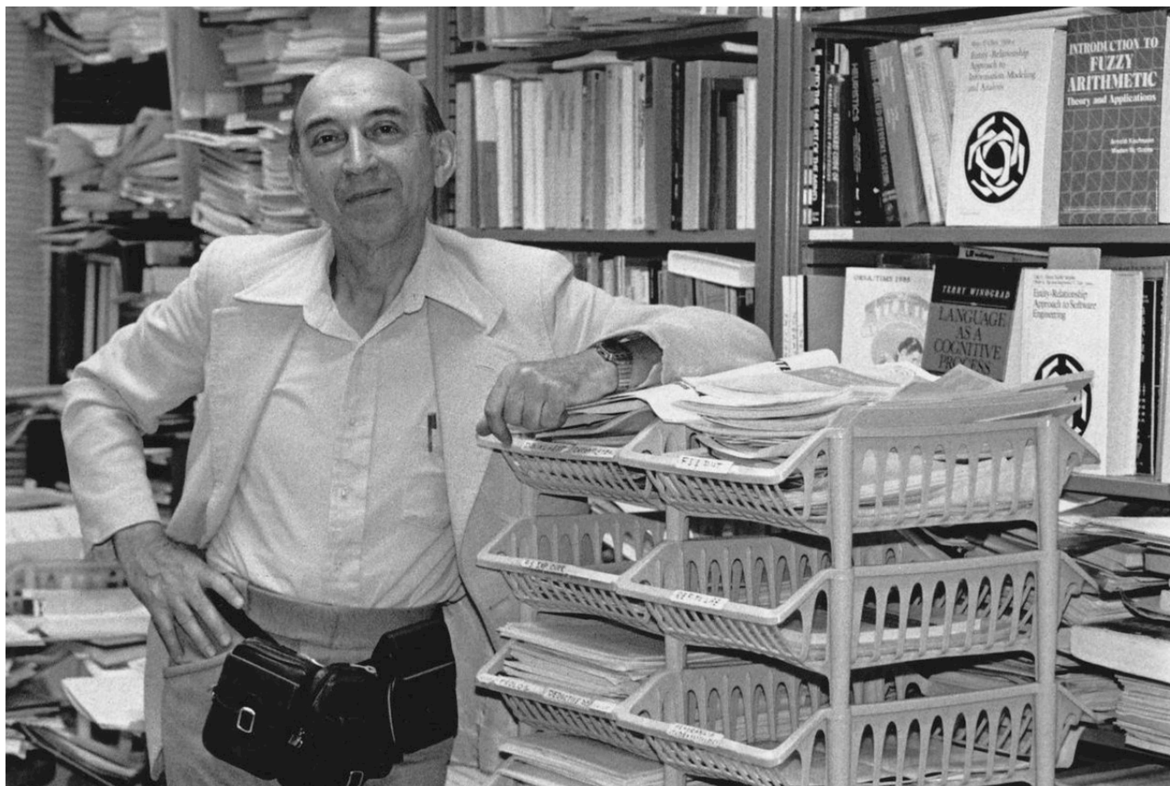
P→ Q			
P \ Q	1	0	*m*
1	1	0	*m*
0	1	1	*m*
m	*m*	*m*	*m*

Sljedeći je korak daljnje proširenje vrijednosnog skupa istinitosti. Prvi je to napravio američki matematičar ***Post*** koji je 1921. godine uveo logički sustav koji je mogao imati *n* vrijednosti istinitosti, a *n* je bio veći ili jednak 2 $(n \geq 2)$. Sličnu su logiku kasnije predložili ***Łukasiewicz*** i ***Tarski***. Łukasiewicz je 1922. godine predložio logiku s $\aleph_0$[53] vrijednosti istinitosti, što znači da vrijednosti istinitosti ima beskonačno u konačnom intervalu vrijednosti između 0 i 1. Sve su ove logike ostale na razini matematike i filozofije, a prvi veći pomak koji je doveo do šire primjene viševaljanih logika u sustavima umjetne inteligencije dogodio se 1965. godine uvođenjem ***teorije neizrazitih*** (engl. ***fuzzy***) ***skupova,*** te kasnije i ***neizrazite*** (engl. ***fuzzy***) ***logike***. Neizrazita teorija skupova i neizrazita logika matematički su temelji ***neizrazitih sustava*** (engl. *Fuzzy Systems*) koji su jedan od važnih dijelova računske inteligencije o kojoj smo govorili u uvodnom poglavlju. Neizraziti sustavi su posebno veliku primjenu našli u sustavima automatskog vođenja, pa su obično iza imena ***inteligentno vođenje*** (engl. *Intelligent Control*) u biti krije tehnologija temeljena na neizrazitoj teoriji skupova i neizrazitoj logici.

[53] $\aleph_0$ – aleph-nula (engl. *aleph-zero, aleph-naught, aleph-null*) iskazuje kardinalnost skupa (broj elemenata) prirodnih brojeva. Kaže se da je kardinalnost skupa prirodnih brojeva $\aleph_0$, a kako prirodnih brojeva ima beskonačno, aleph-nula upravo to i iskazuje.

Torija neizrazitih (fuzzy) skupova i neizrazita (fuzzy) logika

Teoriju neizrazitih skupova (engl. *Fuzzy Set Theory*) uveo je 1965. godine ***Lotfi Aliasker Zadeh*** (1921. – 2017.), profesor na *University of California*, *Berkley,* da bi je kasnije djelomično proširio i na ***neizrazitu logiku*** (engl. *Fuzzy Logic*)[54].



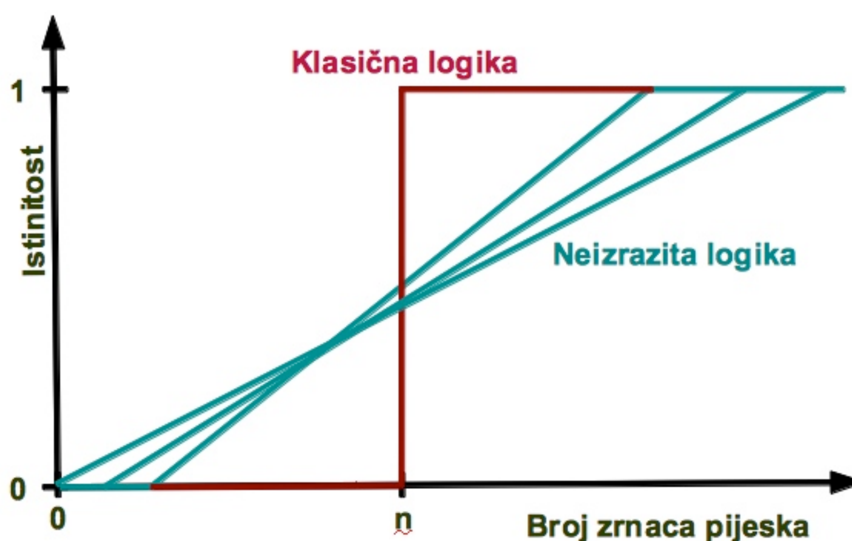
Slika 4-5. Lotfi Aliasker Zadeh, otac neizrazite (fuzzy) logike 1988. godine[55]

Zadeh je neizrazite skupove uveo razmatrajući problem računanja u situacijama ***neodređenosti*** (engl. *Vagueness*), pa se prije svega trebamo upitati što je neodređenost? Neodređenost je vezana uz logički paradoks poznat pod imenom „***Sorites paradoks***", gdje *sorites* dolazi od pridjeva grčke riječi *sōros* (*σωρός*), što znači gomila. Zamislimo gomilu pijeska za koju kažemo „*Ovo je gomila pijeska.".* Maknemo li jedno zrno pijeska iz gomile, je li ona još uvijek gomila? Većina bi ljudi kazala da jest. Ponavljajmo sada isti postupak i uklanjajmo zrno po zrno. Pitanje je koje zrno treba ukloniti da gomila više ne bude gomila. Očito je da nije moguće imenovati baš jedno zrno nakon kojega gomila prestaje biti gomila. Između pojma *„gomila"* i *„negomila"*, ne postoji oštar prijelaz. Ako je zrna malo, to nije gomila, ako ih je puno, to je gomila, a negdje između njih postoji područje neodređenosti. Zadeh je za razrješenje ovog problema uveo pojam *„fuzzy set"* koji hrvatski prevodimo ***neizraziti skup*** koji predstavlja skup koji nema oštre granice između pripadanja i nepripadanja nekog elementa tom skupu (vidi sliku 4-6).

---

[54] Logika i skupovi su čvrsto povezani, pa se tako teorija skupova smatra područjem matematičke logike. 1880. g. *John Venn* je uveo dijagrame, koje po njemu zovemo Vennovi dijagrami, s ciljem vizualizacije logičkih propozicija i relacija. Prije njega nešto slično je uveo i Euler. Eulerovi i Vennovi dijagrami danas su sastavni dio teorije skupova. Kod logike govorimo o istinitosti tvrdnje P, a kod teorije skupova o pripadnosti skupu P. Na primjer, uzmimo tvrdnju *P = „Ivan je visok čovjek."*. U duhu predikatne logike predikat je *visoki_ljudi*, a argument *Ivan* pa predikat *visoki_ljudi(Ivan)* može u klasičnoj logici vratiti vrijednost 0 (laž) ako Ivan nije visok čovjek ili 1 (istina) ako Ivan jest visok čovjek. Kod teorije skupova uzimamo skup *P = „visoki ljudi"* i razmatramo pripadanje jedinke *a = „Ivan"* skupu *„visoki ljudi".* Formalno to možemo napisati pomoću funkcije pripadnosti elementa *a* skupu *P:* $\mu_P(a)$ koja kod standardne teorije skupova može imati vrijednost 0 (ne pripada) ili 1 (pripada), a kod neizrazite teorije skupova bilo koju vrijednost između 0 i 1. Lotfi Zadeh je ideju ***neizrazitosti*** (eng. *fuzzynes*) najprije uveo kroz teoriju skupova, da bi je kasnije interpretirao i logikom.

[55] Za fotografiju Credit to Cindy Manly-Fields, UC Berkley.



Slika 4-6. Ilustracija funkcije istinitosti tvrdnje „Ovo je gomila pijeska". Na apscisnoj osi je broj zrna pijeska, a na ordinati vrijednost istinitosti tvrdnje. Kod klasične logike to može biti 0 ili 1, pa kod jednog broja zrna (n) moramo odlučiti je li sada nastala gomila pijeska ili nije. Kod neizrazitog pristupa vrijednost istinitosti kontinuirano raste. Način rasta funkcije istinitosti može biti subjektivan (nekome je gomila prije, nekome kasnije), ali ne može biti proizvoljan. Nitko neće funkciju istinitosti tvrdnje „Ovo je gomila." definirati padajućom krivuljom. Možda samo Antuntun kod koga je „malo neobičan um" iz pjesme Grigora Viteza „Kako živi Antuntun".

Namjerno ga je nazvao riječju „***fuzzy***" koja u engleskom jeziku nema jedinstveni prijevod, već se može prevoditi kao kovrčav, maljav, nejasan, pomućen, a u kontekstu koji se koristi u umjetnoj inteligenciji najbolje mu odgovara pojam ***neizrazit*** koji i mi koristimo. Suprotni pojam je „***crisp***" kojeg prevodimo ***izrazit*** ili ***jasni***. Cijela se teorija sastoji od toga da postoji beskonačno mnogo stupnjeva pripadanja određenog elementa nekom skupu. Ako on sigurno pripada tom skupu, onda mu je stupanj pripadanja 1, ako sigurno ne pripada, stupanj pripadanja mu je 0, a ako djelomično pripada, onda je stupanj pripadanja bilo koji broj iz intervala od 0 do 1. Drugim riječima, stupanj pripadanja određenog elementa nekom skupu vrijednosti poprima iz zatvorenog intervala [0,1].

Neizrazita logika se može uvesti na više načina. Posebno su zanimljiva dva pristupa (Dubois i Pride, 1980.):

1. *Neizrazitom interpretacijom propozicijske ili predikatne logi*ke, sa ciljem zaključivanja i u situacijama neodređenosti. I dalje imamo predikate, i dalje imamo argumente, samo što predikat ne vraća vrijednost istinitosti iz dvočlanog skupa {0,1}, već iz zatvorenog intervala [0,1]. Obično se naziva ***bazna neizrazita logika*** (engl. *Base* ili *Basic Fuzzy Logic*), a u biti radi se o neizrazitoj interpretaciji *Łukasijewiczeve ∞-valjane logike*.

2. *Uvođenjem lingvističkih vrijednosti istinitosti* na način kako to radimo u prirodnom jeziku. Ovo je Zadehov pristup koji se uobičajeno smatra ***neizrazitom logikom*** (engl. *Fuzzy Logic*), iako ima brojne nedostatke, pa je upitno da li se uopće može smatrati logikom.

Bazna neizrazita logika

U primjeru „*Sorites paradoksa*" sa slike 4-6 vrijednost istinitosti tvrdnje „*Ovo je gomila.*" u ovisnosti o broju zrnaca pijeska koje gomila sadrži je broj između 0 i 1, pri čemu 0 znači potpunu neistinu (laž) tvrdnje, a 1 potpunu istinu tvrdnje. Bilo koji broj između označava istovremeno i neistinitost i istinitost tvrdnje. Na primjer ako je istinitost tvrdnje 0,6 onda je neistinitost ove tvrdnje 1– 0,6 = 0,4 o čemu u nastavku detaljnije govorimo.

U drugom primjeru vratit ćemo se tekstu o predikatnoj logici gdje smo imali predikat *je_tvrd* i argument *kamen* koji je vraćao vrijednost istinitosti 1:

je_tvrd(kamen) = 1

a jezično smo ga interpretirali

„Kamen je tvrd."

I u neizrazitoj logici to ostaje, zato što je kamen stvarno tvrd. Međutim, postavimo li pitanje *je_tvrd(guma)* u standardnoj predikatnoj logici moramo se odlučiti i kazati ili je ili nije, a u neizrazitoj logici kažemo na primjer da je guma tvrda sa ***stupnjem istinitosti*** (engl. *Degree of Truth*) koja ima vrijednost 0,8 kada se radi o tvrdoj gumi, odnosno 0,4 ako se radi o mekoj gumi.

je_tvrd(guma_1) = 0,8

je_tvrd(guma_2) = 0,4

Prvi slučaj mogli bismo jezično interpretirati

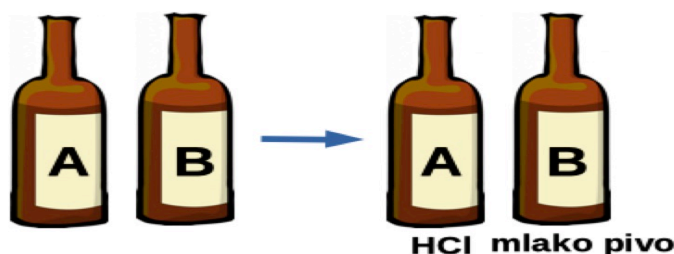
„Guma je prilično tvrda." ili *„Guma je dosta tvrda."*

a drugi

„Guma nije toliko tvrda." ili *„Guma je malo tvrda."*

Kako stupanj istinitosti poprima vrijednosti na zatvorenom intervalu [0;1] često se miješa sa ***stupnjem vjerojatnosti*** (engl. *Degree of Probability*), ali se oni bitno razlikuju. Stupanj vjerojatnosti je vezan s ponavljanjem pokusa, a stupanj istinitosti iskazuje naš, najčešće subjektivni, osjećaj pripadnosti. Zato se ponekad i naziva ***stupanj mogućnosti*** (engl. *Degree of Possibility)*. Pokažimo to primjerom.

Pretpostavimo da ispred sebe imamo dvije grupe od deset boca, grupu A za koju znamo da je ***stupanj vjerojatnosti*** da tekućina u bocama pripada skupini *„tekućina pogodnih za piće"* 0,9 i grupu B za koju znamo da je ***stupanj mogućnosti*** (ili stupanj istinitosti) da tekućina u bocama pripada skupini *„tekućina pogodnih za piće"* također 0,9. Slučajnim odabirom izaberemo jednu bocu iz prve grupe i jednu bocu iz druge grupe. Pitanje je iz koje bismo se boce radije napili? Naravno, iz boce koja ima stupanj mogućnosti da je u njoj tekućina pogodna za piće 0,9. Zašto? Zato što je sigurno da je u svim bocama skupine B tekućina pogodna za piće. Ova grupa ima visoki stupanj istinitosti tvrdnje: *„Ovo je tekućina pogodna za piće."* Tekućine u njima možda nisu idealne, nemaju stupanj istinitosti 1, ali smo prilično sigurni da ni u jednoj od tih boca nije tekućina koja nije pogodna za piće. S druge strane za grupu A koja ima stupanj vjerojatnosti 0,9 da je u boci *„tekućina pogodna za piće"*, šansa nam je 1:10 da naletimo na bocu u kojoj nije tekućina pogodna za piće. Stupanj mogućnosti iskazuje stupanj *„koliko je tekućina pogodna za piće"*, a vjerojatnost *„je li tekućina pogodna za piće"*. Pojam *„nije pitka"* je puno opasniji od pojma *„ne tako dobra za piće"*. U grupi A, i to možda baš u boci koju smo odabrali, može biti i otrovna tekućina, na primjer klorovodična kiselina, a u grupi B, u najgorem slučaju može biti nešto što se procjenitelju nije svidjelo, na primjer mlako pivo, pa je zato dao stupanj mogućnosti 0,9 (slika 4-7).



Slika 4-7. Vjerojatnost i mogućnost – za skupinu A kažemo da je vjerojatnost da je u boci tekućina pogodna za piće 0,9, a za skupinu B da je mogućnost da je u boci pitka tekućina 0,9. Iz koje biste se skupine radije napili?

Ovaj primjer ilustrira kako neizrazitost i vjerojatnost u osnovi nose dvije potpuno različite vrste informacija. ***Neizrazitost*** *iskazuje pripadnost ili sličnost objekta neprecizno definiranom svojstvu, a* ***vjerojatnost*** *daje relativnu učestalost pojave tog svojstva kod analiziranih objekata.* Pridruživanje vrijednosti istinitosti neizrazitom sudu rezultat je iskustva i intuicije procjenitelja, a vrijednost vjerojatnosti rezultat je ponavljanja pokusa.

Kao ilustraciju pojma neizrazitosti pogledajmo i drugi primjer. Natko živi u stanu koji ima samo dvije prostorije: kuhinju i sobu. Pitamo se gdje je Natko? Je li on u sobi ili kuhinji? Ako kažemo da je vjerojatnost da je Natko u sobi *0,5* onda je on sigurno ili u sobi ili u kuhinji. Zavirimo li deset puta Natku u stan, on će pet puta biti u sobi, a pet puta biti u kuhinji. A što ako je Natko točno na vratima između sobe i kuhinje? Vjerojatnost ne može uhvatiti taj slučaj s obzirom na to da on nije ni u sobi ni u kuhinji, već točno između njih. Sa stupnjem mogućnosti ili stupnjem istine hvatamo baš taj slučaj kada kažemo da je mogućnost da je Natko u sobi 0,5, ali isto tako i da je mogućnost da je Natko u kuhinji *0,5*. Zavirimo li deset puta Natku u stan, on će svih deset puta biti na vratima između kuhinje i sobe, znači na neki način jednako i u kuhinji i u sobi.

Složeni izrazi ovakve interpretacije ***neizrazite logike*** formiraju se povezivanjem osnovnih neizrazitih izraza logičkim operacijama. Logičke operacije su iste kao kod propozicijske i predikatne logike, ali, naravno, imaju potpuno drugačije tablice istinitosti. I u ovom je dijelu neizrazita logika ostala dosljedna sebi, na način da ne postoji samo jedna neizrazita logika, već nekoliko njih, koje se razlikuju po tome na koji se način definiraju temeljni logički operatori konjunkcije i disjunkcije (negacija se u svima njima definira na isti način).

Neka su $P^*(X)$ i $Q^*(X)$ dva predikata neizrazite logike koji imaju vrijednosti istinitosti p i q. Oznakom * naglašavamo da se radi o predikatima neizrazite logike.

$P^*(X) = p \;,\; Q^*(X) = q \;\; ; p, q \in [0,1]$

Na primjer, možemo uzeti predikat *biti_u_kuhinji* i *biti_u_sobi*, a argument je Natko.

biti_u_kuhinji(Natko)=0,3

biti_u_sobi(Natko)=0,6

Stupnjevi mogućnosti iskazuju naš subjektivni osjećaj gdje je Natko poznavajući navike njegovog ponašanja.

„Mogućnost da je Natko u kuhinji je 0,3 , a mogućnost da je Natko u sobi je 0,6.”

Negacija $\neg$ se u svim neizrazitoj logici definira izrazom

$\neg P^*(X) = 1 - p$

Za naš primjer vrijedi:

$\neg$ *biti_u_kuhinji(Natko)= 1 – 0,3 = 0,7*

$\neg$ *biti_u_sobi(Natko)= 1 – 0,6 = 0,4*

odnosno

„Mogućnost da Natko nije u kuhinji je 0,7, a mogućnost da Natko nije u sobi je 0,4. ”

Operator konjunkcije (veznik *i*) definira se matematičkom binarnom operacijom koja se naziva ***triangularna*** ili ***T-norma***. T-norma je funkcija sa Kartezijevog produkta zatvorenih intervala *[0,1]* na isti taj zatvoreni interval:

$$T: [0;1] \times [0;1] \rightarrow [0;1] \qquad (4\text{-}1)$$

sa svojstvima

KOMUTATIVNOSTI	$T(a\,;\,b) = T(b\,;\,a)$
MONOTONOSTI	$T(a\,;\,b) \leq T(c\,;\,d)$, ako je $a \leq c$ i $b \leq d$
ASOCIJATIVNOSTI	$T(a\,;\,T(b;\,c)) = T(T(a\,;\,b);\,c)$
NUL ELEMENT	$T(a\,;\,0) = 0$

ELEMENT IDENTITETA $T(a\ ;\ 1) = a$

Svaka T-norma ima svoju ***konormu*** koja se naziva ***S-norma*** a dobije se iz T-norme izrazom

$$S(a\ ;\ b) = 1 - T(1-a\ ;\ 1-b) \tag{4-2}$$

S-norma također zadovoljava svojstva komutativnosti, monotonosti i asocijativnosti, a kod nje je nul element 1, a element identiteta 0. U neizrazitoj logici S-norma služi za definiranje operatora disjunkcije (veznik ili). Postoji puno različitih T-normi i S-normi i sve one mogu poslužiti kod definiranja tablica istinitosti operacija neizrazite logike. U praksi je najpopularnija i možda najjednostavnija neizrazita logika koja koristi operaciju minimuma (min) kao T-normu i operaciju maksimuma (max) kao S-normu.

$$T(a\ ;\ b)= min(a\ ;\ b) \tag{4-3}$$

$$S(a\ ;\ b) = max(a\ ;\ b) \tag{4-4}$$

Ova se neizrazita logika zbog toga i naziva ***min-max neizrazita logika*** ili najčešće samo ***neizrazita logika.*** Drugi primjer neizrazite logike je ***Lukasiewiczeva neizrazita logika*** kod koje vrijedi:

$$T(a\ ;\ b)= max(0\ ;\ a+b-1) \tag{4-5}$$

$$S(a\ ;\ b) = min(a+b,1) \tag{4-6}$$

a ponekad se koristi i ***produkt-suma neizrazita logika*** kod koje je:

$$T(a\ ;\ b)= a \cdot b \tag{4-7}$$

$$S(a\ ;\ b) = a + b - a \cdot b \tag{4-8}$$

U nastavku razmatramo samo min-max neizrazitu logiku koju ćemo dalje jednostavno nazivati samo neizrazita logika. Tablica 4-5 daje istinitosti osnovnih logičkih operatora neizrazite logike.

Tablica 4-5. Tablica istinitosti tvrdnji povezanih različitim logičkim operatorima neizrazite min-max logike

P^*	Q^*	$P^*\dot{}Q^*$	$P^* \vee Q^*$	$Q^* \rightarrow P^*$	P^*
p	*q*	*max(p;1-p;q;1-q)*	*max(p;q)*	*max(p;1-q)*	*p*
P^*	Q^*	$P^* \rightarrow Q^*$	Q^*	$P^* \leftrightarrow Q^*$	$P^* \wedge Q$
p	*q*	*max(1-p;q)*	*q*	*min(max(1-p;q); max(p;1-q))*	*min(p;q)*
P^*	Q^*	$P^*\vert Q^*$	P^* ex Q^*	$\neg Q^*$	$Q^* \rightsquigarrow P^*$
p	*q*	*max(1-p;1-q)*	*max(min(1-p;q); min(p;1-q))*	*1-q*	*min(p;1-q)*
P^*	Q^*	$\neg P^*$	$P^* \rightsquigarrow Q^*$	$P^* \downarrow Q^*$	$P^*\mathring{}Q^*$
p	*q*	*1-p*	*min(1-p;q)*	*min(1-p;1-q)*	*min(p;1-p; q;1-q)*

U tablici je navedena standardna definicija operacije implikacije, međutim u praksi se najviše koristi još jednostavnija definicija implikacije koja je poznata kao ***Mamdanieva implikacija***:

$$P^*(X)\rightarrow Q^*(X) = min\ (p\ ;\ q) \tag{4-9}$$

Ime je dobila po profesoru ***Ebrahimu Mamdaniu*** (1942.-2010.) koji je tijekom rada na Queen Mary Collegu u Londonu 1977. godine objavio prvu praktičnu primjenu neizrazite logike u vođenju složenog procesa koji je uspješno do tada mogao voditi samo čovjek. Na taj način uveo je pojam

neizrazitog vođenja (engl. *Fuzzy Control*), komercijalno sigurno najuspješnijeg proizvoda inteligentnih tehnologija.

Vratimo se Natku i njegovom boravku u stanu. Istinitost tvrdnji: *„Natko je u kuhinji i u sobi."* i *„Natko je u kuhinji ili u sobi."* računamo u duhu min-max neizrazite logike

$P^*(X) \wedge Q^*(X) = min(p\ ;\ q)$

$P^*(X) \vee Q^*(X) = max(p\ ;\ q)$

biti_u_kuhinji(Natko) $\wedge$ *biti_u_sobi(Natko) = min (0,3 ; 0,6) = 0,3*

biti_u_kuhinji(Natko) $\vee$ *biti_u_sobi(Natko) = max (0,3 ; 0,6) = 0,6*

Zadehova neizrazita logika

Zadehova neizrazita logika povezana je s postupkom koji se zove ***računanje s riječima*** (engl. *CWW – Computing With Words*). Zadeh je predložio ideju pridavanja matematičkog značenja riječima i rečenicama prirodnog jezika korištenjem neizrazitih skupova. Računanje s riječima omogućava uspješnu matematičku formalizaciju znanja čovjeka iskazanu riječima i rečenicama prirodnog jezika pogodnu za primjenu u sustavima automatskog zaključivanja.

Kod ove neizrazite logike istinitost tvrdnje *P* određujemo lingvističkim vrijednostima istinitosti $\tau(P)$ koji pripadaju prebrojivom skupu *TV* (kratica dolazi od engl. riječi *Truth Values*), na primjer:

TV = {istinito, lažno, neistinito, vrlo istinito, ne vrlo istinito, prilično istinito, ... }

Svaka od ovih lingvističkih vrijednosti istinitosti se definira neizrazitim skupom definiranom preslikavanjem sa zatvorenog intervala [0;1] na taj isti interval:

$$\mu_{\tau(P)}: [0; 1] \rightarrow [0; 1] \qquad (4\text{-}10)$$

Pri tome je dovoljno definirati samo neizraziti skup lingvističke vrijednosti *„istinito"*, a ostale lingvističkih vrijednosti istinitosti računamo koristeći odgovarajuću matematičku formu za modeliranje dodanih riječi *„ne", „vrlo", „prilično"* itd. Zadeh ove dodane riječi naziva ograde (engl. *Hadges*), a definira ih pravilima proračuna (engl. *Rules of Computation*). Za pridružna funkcija pojma *istinito* $\mu_{istinito}(X)$, $X \in [0;1]$, ostale vrijednosti istinitosti iz skupa *TV* definiramo izrazima:

$$\mu_{lažno}(X) = \mu_{istinito}(1 - X) \qquad (4\text{-}11)$$

$$\mu_{neistinito}(X) = 1 - \mu_{istinito}(X) \qquad (4\text{-}12)$$

$$\mu_{vrlo_istinito}(X) = (\mu_{istinito}(X))^2 \qquad (4\text{-}13)$$

$$\mu_{prilično_istiniti}(X) = (\mu_{istinito}(X))^{1/2} \qquad (4\text{-}14)$$

Treba uočiti razliku između pojmova *„lažno"* i *„neistinito"*. *„Lažno"* je vrijednost istinitosti negacije tvrdnje *P* koju zapisujemo $\tau(\neg P)$, dok je *„neistinito"* negacija istinitosti tvrdnje P koja zapisujemo $\neg\tau(P)$.

Pogledajmo primjer s diskretnim neizrazitim skupovima istinitosti koji se u praktičnim primjenama više koriste. Za razliku od kontinuiranih neizrazitih skupova istinitosti koji kao podršku Ω imaju kontinuirani interval *[0;1]*, kod diskretnih neizrazitih skupova istinitosti vrijednosti su ograničene na diskretni skup podrške. U našem primjeru on će imati 11 vrijednosti $\Omega_D = \{0\ ;\ 0,1\ ;\ 0,2\ ;\ 0,3\ ;\ 0,4\ ;\ 0,5\ ;\ 0,6\ ;\ 0,7\ ;\ 0,8\ ;\ 0,9\ ;\ 1\}$. Lingvističke vrijednosti istinitosti sada definiramo preslikavanjem:

$$\mu_{\tau}: \Omega_D \rightarrow [0; 1] \qquad (4\text{-}15)$$

Pri tome koristimo poseban način zapisivanja. Pogledajmo ga na primjeru definiranja pojma *„istinito"* diskretnim neizrazitim skupom:

$$\mu_{istinito}(X) = 0,2/0,6 + 0,4/0,7 + 0,6/0,8 + 0,8/0,9 + 1/1$$

U ovom se zapisu ne radi se o razlomcima ili operacijama dijeljenja i zbrajanja, već o simboličkom zapisu diskretnog neizrazitog skupa. U brojniku je vrijednost istinitosti, a u nazivniku element diskretne podrške kojem je ta vrijednost pridijeljena. Uobičajeno je da se u ovakvom zapisu ne navode one vrijednosti podrške za koje je vrijednost pridružne funkcije *0*. Interpretiramo ga na slijedeći način. Istinitost elementa podrške *1* pripada pojmu *„istinito”* sa stupnjem pripadnosti *1*, elementa *0,9* sa stupnjem pripadnosti *0,8* itd., a svi elementi podrške manji ili jednak od *0,5* ne pripadaju pojmu *„istinito”*. Sada definiramo i ostale vrijednosti istinitosti:

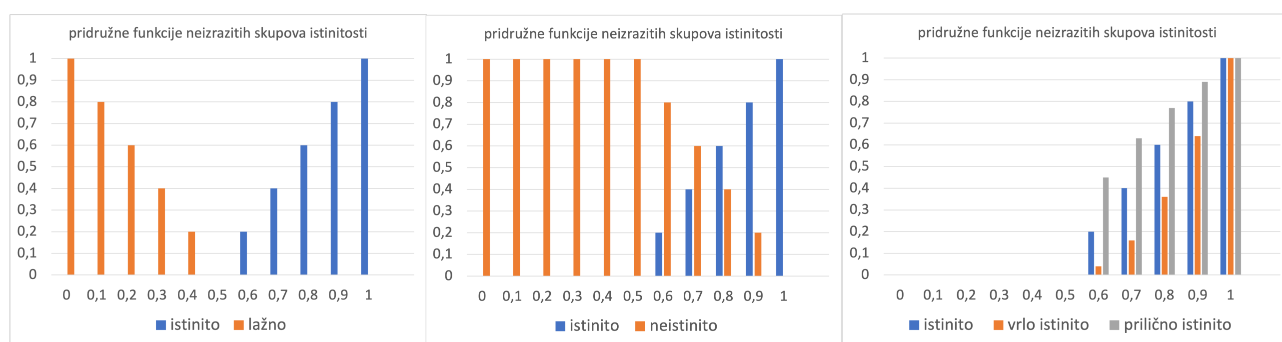
$$\mu_{lažno}(X) = 1/0 + 0{,}8/0{,}1 + 0{,}6/0{,}2 + 0{,}4/0{,}3 + 0{,}2/0{,}4$$

$$\mu_{neistinitoi}(X) = 1/0 + 1/0{,}1 + 1/0{,}2 + 1/0{,}3 + 1/0{,}4 + 1/0{,}5 + 0{,}8/0{,}6 + 0{,}6/0{,}7 + 0{,}4/0{,}8 + 0{,}2/0{,}9 + 0/1$$

$$\mu_{vrlo_istinito}(X) = 0{,}04/0{,}6 + 0{,}16/0{,}7 + 0{,}36/0{,}8 + 0{,}64/0{,}9 + 1/1$$

$$\mu_{prilično_istiniti}(X) = 0{,}45/0{,}6 + 0{,}63/0{,}7 + 0{,}77/0{,}8 + 0{,}89/0{,}9 + 1/1$$

Slika 4-8 prikazuje pridružne funkcije ovih vrijednosti istinitosti.



Slika 4-8. Pridružne funkcije diskretnih vrijednosti istinitosti

Pogledajmo sada primjere korištenja logičkih operatora negacije. Uzmimo tvrdnju *P = „Natko crta dobro.”* čija je istinitost *„vrlo istinito”* definirana pridružnom funkcijom:

$$\tau(P) = 0{,}4/0{,}6 + 0{,}16/0{,}7 + 0{,}36/0{,}8 + 0{,}64/0{,}9 + 1/1$$

Da je istinitost ove tvrdnje bila *„ne vrlo istinito”* tada bi njena pridružna funkcija bila:

$$\neg\tau(P) = (1\text{-}0)/0 + (1\text{-}0)/0{,}1 + (1\text{-}0)/0{,}2 + (1\text{-}0)/0{,}3 + (1\text{-}0)/0{,}4 + (1\text{-}0)/0{,}5 + (1\text{-}0{,}4)/0{,}6 + (1\text{-}0{,}16)/0{,}7 +$$
$$+ (1\text{-}0{,}36)/0{,}8 + (1\text{-}0{,}64)/0{,}9 + (1\text{-}1)/1 =$$
$$= 1/0 + 1/0{,}1 + 1/0{,}2 + 1/0{,}3 + 1/0{,}4 + 1/0{,}5 + 0{,}6/0{,}6 + 0{,}84/0{,}7 + 0{,}64/0{,}8 + 0{,}36/0{,}9$$

što je potpuno drugačije od istinitosti tvrdnje *¬P = „Natko ne crta dobro.”* čija bi pridružna funkcija bila:

$$\tau(\neg P) = 0{,}4/(1\text{-}0{,}6) + 0{,}16/(1\text{-}0{,}7) + 0{,}36/(1\text{-}0{,}8) + 0{,}64/(1\text{-}0{,}9) + 1/(1\text{-}1) =$$
$$= 0{,}4/0{,}4 + 0{,}16/0{,}3 + 0{,}36/0{,}2 + 0{,}64/0{,}1 + 1/0 = 1/0 + 0{,}64/0{,}1 + 0{,}36/0{,}2 + 0{,}16/0{,}3 + 0{,}4/0{,}4$$

Kod primjene ostalih logičkih operatora koristi se ***princip ekstenzije*** (engl. *Extension principle*) temeljen na Tablici 4-5, čiji detaljni opis prelazi okvire ovog udžbenika, pa upućujemo čitaoce na literaturu[56].

Spomenimo još i to da je jedna od razlika između beskonačno valjanih logika predloženih početkom 20. stoljeća i Zadehove neizrazite logike u tome što, na primjer, Łukasiewiczeva beskonačno valjana logika poznaje samo dva kvantifikatora kao i klasična logika (svi i neki), a neizrazita logika uvodi, bar teorijski, beskonačno mnogo kvantifikatora na način jednak onome na koji su kvantifikatori definirani i u prirodnom jeziku. Jezik poznaje veliki broj kvantifikatora – na primjer, kada u jeziku neko

[56] Na primjer rad Abdul Khaliq i Amais Ahmad „Fuzzy Logic and Approximate Reasoning" - https://www.diva-portal.org/smash/get/diva2:832139/FULLTEXT01.pdf

svojstvo želimo pridijeliti određenoj grupi jedinki, onda se može kazati: *„skoro svi", „jako puno", „vrlo malo", „ne više od 100", „velika većina", „teško bilo koji", „skoro ni jedan"* itd. Njihovo značenje je svima dobro poznato, a pitanje je kako ovakve izraze matematički formalizirati. Koristi se isti princip kao kod definiranja vrijednosti istinitosti temeljen na Zadehovom računanju s riječima.

Ovako definirana neizrazita logika ne zadovoljava brojna svojstva koja bi formalna logika morala zadovoljavati, pa se po nekim autorima i ne može smatrati logikom. Osnovni problem je taj što kod proračuna istinitosti složene tvrdnje sastavljene iz više tvrdnji povezanih logičkim operatorima ne dobije neka od vrijednosti iz skupa dozvoljenih vrijednosti istinitosti *TV*. Najčešće se dobije neizraziti skup istinitosti koji se ne poklapa niti s jednim od dozvoljenih neizrazitih skupova vrijednosti istinitosti iz skupa *TV*. I sam Zadeh je bio svjestan ovog problema, pa je predložio postupak ***lingvističke aproksimacije*** (engl. *Linguistic Approximation*) kojim se dobiveni neizraziti skup preslikava u neku od vrijednosti istinitosti iz skupa *TV*. Međutim i bez zadovoljavanja matematičkog formalizma neizrazita logika je pronašla brojne praktične primjene u različitim područjima o čemu detaljnije govorimo u poglavlju o neizrazitom zaključivanju i rasuđivanju.

Vjerojatnosne logike

Kao zadnju logiku, možda jednu od najraširenijih, barem što se tiče upotrebe u dijagnostičkim ekspertnim sustavima, navest ćemo tzv. ***vjerojatnosnu logiku***, odnosno s njom povezano ***vjerojatnosno zaključivanje***.

Do sada smo uglavnom razmatrali metode kod kojih se pretpostavlja da je tvrdnja istinita ili lažna, odnosno kod neizrazite logike negdje između istinite i lažne, a nismo uzeli u obzir mogućnost da nešto bude ***„vjerojatno istinito"***. Uvođenje vrijednosti istinitosti vodi prema vjerojatnosnom zaključivanju i uglavnom se koristi kod tri tipa problema:

- kada su ulazne informacije slučajne – npr. distribucija oboljenja ljudi za vrijeme epidemije
- vanjski svijet nije toliko slučajan, ali nemamo uvijek dostupne sve informacije, npr. izvjesnost uspjeha ako neki lijek koristimo za liječenje neke bolesti i
- svijet izgleda slučajan zato što ga nismo opisali dovoljno detaljno, npr. prepoznavanje pisanog rukopisa.

Tipičan primjer je ekspertni sustav medicinske dijagnostike. Nikada se ne može tvrditi da su neki simptomi apsolutno presudni za neku bolest, ali postoje vjerojatnosti (uvjetne vjerojatnosti) da je neka hipoteza ispravna ako je prisutan određeni uvjet.

Vjerojatnosne logike su se uvele na različite načine, od jednostavne pretpostavke da su vrijednosti istine vjerojatnosti, pa do kompleksnijih sustava kod kojih se vjerojatnosti ne povezuju direktno s tvrdnjama. Ipak, najzanimljivija primjena vjerojatnosti je u ***vjerojatnosnom zaključivanju*** koje se zasniva na ***Bayesovom teoremu***. Osmislio ga je engleski matematičar, filozof i teolog ***Thomas Bayes*** (1701. – 1761.), a zanimljivo je da je prvi put javno prezentiran tek 1763. godine, dvije godine nakon Bayesove smrti. O samom zaključivanju govorimo u jednom od sljedećih poglavlja, a ovdje naglasimo samo to da se znanje potrebno za vjerojatnosno zaključivanje pohranjuje u vidu uvjetnih vjerojatnosti. Poznata znanja vezana su s određenom hipotezom i uvjetima koji nju potkrepljuju. Ako su:

$P(E\,|\,H_i)$ – vjerojatnosti da ćemo opaziti zadovoljenje uvjeta E ako je istinita hipoteza H_i (uvjetna vjerojatnost)

$P(H_i)$ – vjerojatnost da je hipoteza H_i istinita ako nemamo nikakvih opažanja (*a priori* vjerojatnost od H_i),

$P(E)$ – vjerojatnost da je uvjet E istinit nezavisno od hipoteze H_i (*a priori* vjerojatnost od E)

$P(E \wedge H_i)$ – vjerojatnost da su i uvjet E i hipoteza H_i istiniti (združena vjerojatnost)

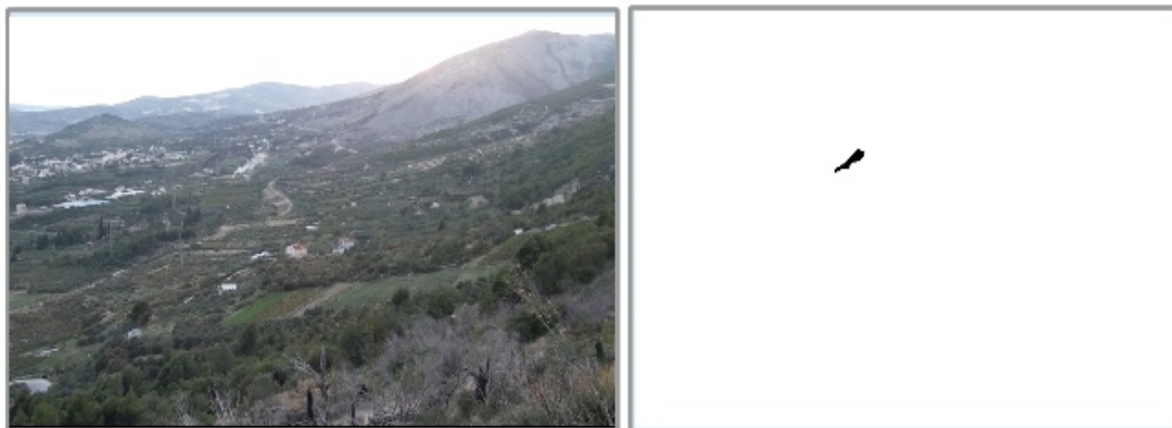
Bayesov teorem daje način kako izračunati:

$P(H_i\,|\,E)$ – vjerojatnost da je hipoteza H_i istinita ako je zadovoljen uvjet E (*a posteriori* vjerojatnost od H_i):

$$P(H_i|E) = \frac{P(E \wedge H_i)}{\mathrm{P(E)}} = \frac{P(E|H_i)\cdot P(H_i)}{\mathrm{P(E)}} = \frac{P(E|H_i)\cdot P(H_i)}{\sum_i P(E|H_i)\cdot P(H_i)} \tag{4-16}$$

Uvjetne vjerojatnosti dobivamo na temelju stvarnog ponavljanja eksperimenta, za razliku od stupnja mogućnosti neizrazite logike do kojih dolazimo na temelju subjektivne procjene eksperta.

Pogledajmo primjer proračuna uvjetne vjerojatnosti. Slika 4-9 prikazuje sliku na kojoj se nalazi dim šumskog požara i ručno segmentirana slika na kojoj je dim izdvojen od strane čovjeka procjenitelja.



Slika 4-9. Ulazna slika na kojoj se vidi dim šumskog požara i ručno segmentirana slika na kojoj je samo dim izdvojen od strane čovjeka procjenitelja

Neka je hipoteza H_i *„Na slici je dim šumskog požara.*", a uvjet *E* pojava određene sive nijanse, npr. srednje vrijednosti sive komponente *rgb(127,127,127)*. Da bismo izračunali uvjetnu vjerojatnost $P(E|H_i)$ treba nam *n* ovakvih različitih slika na kojima je dim šumskog požara i na kojima je dim ručno segmentiran. Sljedeći je korak da za svih *n* slika, ali samo za onaj dio slike koji pripada dimu, prebrojimo piksele koji imaju boju *rgb(127,127,127)*. Ako ukupno na svim slikama imamo *m* piksela koji pripadaju dimu, a *k* piksela ima boju *rgb(127,127,127)*, uvjetna vjerojatnost da će neki piksel imati boju *rgb(127,127,127),* ako pripada dimu je $P(E|H_i) = k/m$. Detaljnije ćemo o zaključivanju korištenjem vjerojatnosnog zaključivanja i rasuđivanju govoriti u poglavlju 5..

Nemonotone logike

Na kraju objasnimo još i pojam ***nemonotonih logika*** (engl. *Non-monotonic Logic*)**.** Da bismo ih shvatili, trebamo objasniti pojam ***monotonosti zaključivanja*** (engl. *Monotonicity of Entailment*). Tradicionalni sustavi zasnovani na predikatnoj logici su monotoni u smislu nove činjenice ne dovode u pitanje istinitost neke od već postavljenih tvrdnji. Na primjer istinitost klasičnog načina silogističkog zaključivanja *„Ivan je muškarac. Svi muškarci nekad su bili dječaci. Zbog toga Ivan je nekad bio dječak.*" se ne mijenja ako uvedemo novu tvrdnju: *„Ivan je muškarac. Svi muškarci nekad su bili dječaci. Krave daju mlijeko. Zbog toga Ivan je nekad bio dječak.*". Prednost ovakvog sustava je ***konzistentnost***, što znači da dodavanjem nove tvrdnje nije potrebno provjeravati jesu li prethodne tvrdnje s njom konzistentne i nije potrebno pratiti koje su sve tvrdnje već dokazane. Nedostaci monotonih logika su što ne mogu rješavati situacije kod kojih:

- imamo nekompletne informacije
- imamo promjenjive situacije ili
- koristimo pretpostavke pri rješavanju,

a to su upravo problemi koji se javljaju pri rješavanju stvarnih životnih zadataka. Rijetko kada su nam dostupne sve informacije potrebne za rješavanje određenih zadataka, međutim, iako informacije nedostaju, uvijek postoji mogućnost pogađanja, ako ne postoje dokazi o protivnom.

Primjer: Pretpostavimo da smo pozvani na ručak u kuću u koju prvi put idemo. U prolazu prolazimo pored prodavaonice cvijeća. Pitanje je hoćemo li domaćici kupiti cvijeće? Na ovu odluku utječe velik broj činjenica, ali mi ćemo se vjerojatno držati općenitog pravila *da većina ljudi voli cvijeće*, pa

pretpostavljamo da i domaćica kod koje idemo također voli cvijeće, te ćemo cvijeće i kupiti. Međutim, ako već na vratima osoba počne kihati zato što je alergična na cvijeće, mi nužno moramo mijenjati istinitost prethodne tvrdnje, ali ne samo nje nego i svih tvrdnji koje su s njom povezane.

Ovakvo zaključivanje na temelju najvjerojatnijih pretpostavki naziva se ***abduktivno zaključivanje*** o kojem više govorimo u Poglavlju 5. Primjer tvrdnji koje ga održavaju je *„većina ljudi voli cvijeće"*, *„većina pasa ima rep"* itd., pa ako nemamo dokaza u protivnost te tvrdnje, mi je prihvaćamo i po njoj se rukovodimo. U praksi nam je često potreban baš ovakav način zaključivanja, a posebno u situacijama kada imamo nepotpune informacije, promjenjive situacije ili situaciju kada za pronalazak potpunog rješenja trebamo postaviti pretpostavke o parcijalnim rješenjima. ***Nemonotono zaključivanje*** (engl. *Non-monotonic Reasoning*) je zaključivanje kod kojeg osim abduktivnog zaključivanja imamo i mogućnost mijenjanja istinitosti tvrdnji za koje smo ustanovili da više nisu istinite. Puno ga je teže realizirati nego monotono zaključivanje, prije svega zato što uklanjanje neke od tvrdnji može utjecati na niz drugih tvrdnji koje tada isto tako treba korigirati. Zbog toga je svaki put, osim izvedene tvrdnje, nužno spremiti i cjelokupni dokaz, odnosno barem slijed tvrdnji na temelju kojih smo izveli danu tvrdnju.

Primjer sustava pomoću kojeg se može implementirati nemonotono zaključivanje je ***sustav održavanja istinitosti*** (engl. *TMS – Truth Maintenance System*). Uveo ga je 1979. godine ***Jon Doyle*** sa MIT-a. TMS nije sustav zaključivanja, već sustav koji podržava zaključivanje na način da provjerava konzistentnost baze znanja. Svaka od tvrdnji koja se u TMS-u naziva čvor može biti u jednom od dva stanja:

IN – kada za nju ***vjerujemo*** da je istinita

OUT – kada ***ne vjerujemo*** da je istinita, bilo zbog toga što nema razloga da u nju vjerujemo bilo zbog toga što trenutno nisu važeći razlozi da u nju vjerujemo.

Osim toga, svakom čvoru pridodajemo listu provjera pomoću kojih utvrđujemo valjanost (istinitost) čvora. Jedna od takvih provjera je tzv. ***„Support List***" *(SL)*, a temelji se na tome da i svakom pojedinom čvoru (tvrdnji) pridodamo listu čvorova na osnovi koje zaključujemo o valjanosti te tvrdnje i koji su sada u IN-listi, te listu čvorova na osnovi koje bismo zaključivali lažnost, ali koji su trenutno u OUT-listi. Pogledajmo primjer.

Čvor (tvrdnja) koji ima obje liste prazne naziva se ***premisa***. Tipičan primjer je:

`(1) zima je` *`(SL (.) (.))`*

To je čista tvrdnja koja je sada u generalnoj IN listi. Neka je druga tvrdnja:

`(2) hladno je` *`(SL (1) (.))`*

Ona u svoj listi na temelju koje zaključujemo njenu istinitost ima tvrdnju (1). Ovdje je važno naglasiti da mi tvrdnje uključujemo nezavisno o TMS. TMS samo provjerava konzistentnost. Pretpostavimo sada da imamo i treću tvrdnju:

`(3) toplo je`

koja nema ni jednu od lista zato što je trenutno u generalnoj OUT listi. Međutim, ako je (3) na popisu trebamo izmijeniti (2) na način

`(2) hladno je` *`(SL (1) (3))`*

Ovakva tvrdnja naziva se ***pretpostavka***. Ona je istinita ako je tvrdnja

`(1) zima je` - u IN-listi i tvrdnja
`(3) toplo je` - u OUT-listi.

Međutim, ako tvrdnja (3) uđe u IN listu

`(3) toplo je` *`(SL (.) (.))`*

tada tvrdnja (2) više nije istinita. Znači možemo imati dvije moguće situacije. Prva situacija:

`IN ((1),(2))  OUT ((3))`
`(1) zima je` *`(SL ( ) ( ))`* – istinita
`(2) hladno je` *`(SL(1) (3))`* – istinita

`(3) toplo je` – lažna

ili druga situacija:

`IN ((1), (3))  OUT ((2))`

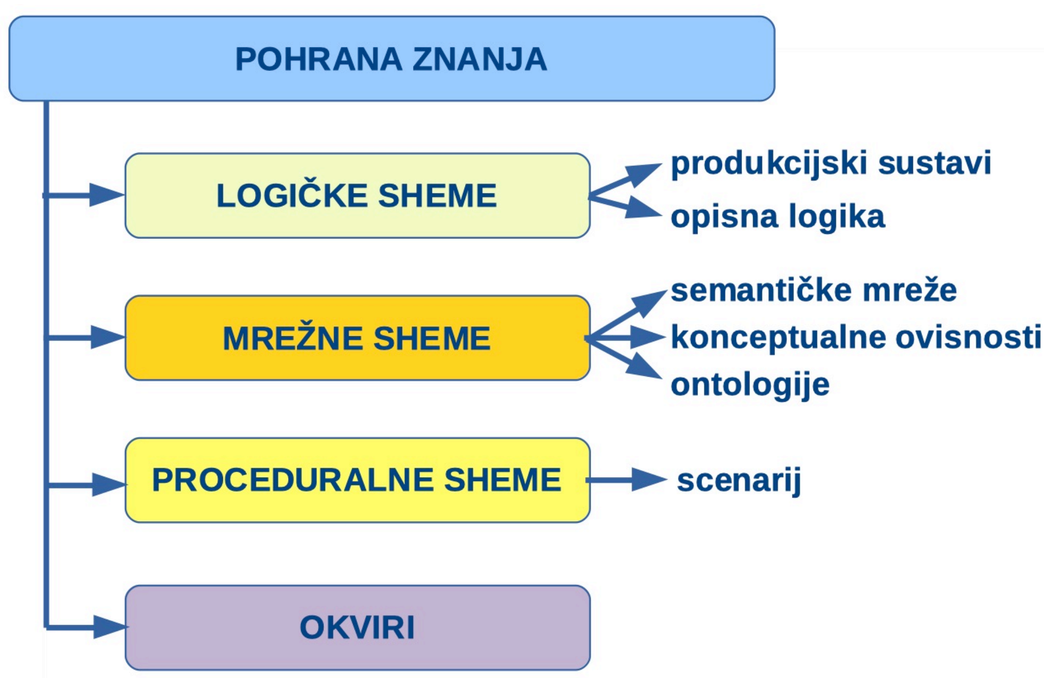
`(1) zima je` *`(SL ( ) ( ))`* – istinita
`(2) hladno je` *`(SL(1) (3))`* – lažna
`(3) toplo je` *`(SL( ) ( ))`* – istinita

TMS uvijek provjerava sve pretpostavke. Međutim, ima slučajeva kada je broj pretpostavki velik pa provjera konzistentnosti oduzima mnogo vremena. Zbog toga je razvijen i ***sustav održavanja istinitosti temeljen na pretpostavkama*** (engl. *ATMS - Assumption Truth Maintenance System*) kod kojega se na osnovi pretpostavki višeg reda provjeravaju samo pojedine pretpostavke iz baze znanja.

4.4 Strukturna pohrana znanja

Kognitivna psihologija razlikuje dva tipa znanja: deklarativno i proceduralno. **Deklarativno znanje** je statičko znanje koje se odnosi na činjenice, njihove atribute i relacije među njima, na primjer C je treće slovo abecede, krava ima 4 noge, krava je sisavac i slično. **Proceduralno znanje** je dinamičko znanje koje se odnosi na to kako se nešto događa ili kako se nešto postiže, na primjer postupak kako se kuha čaj.

Znanje se formalno pohranjuje u baze znanja, strukturne forme organizirane tako da se lako može doći do bilo kojeg elementa unesenog znanja, ali isto tako i jednostavno provoditi rasuđivanje, traženje zaključaka, novoga znanja koje nije već eksplicitno prisutno u bazi znanja. Strukturno organiziranje baza znanja jednim je dijelom preuzelo podjelu znanja na tipove koje predlaže kognitivna psihologija, pa su razvijene različite deklarativne i proceduralne sheme za pohranu znanja, ali su uz njih predložene i neke druge strukture. Sve ih možemo grubo svrstati u četiri grupe (slika 4-10):



Slika 4-10. Neke od shema za strukturnu pohranu znanja

- logičke sheme za prikaz znanja
- deklarativne (mrežne) sheme za prikaz znanja

- proceduralne sheme za prikaz znanja i
- okvire.

U ***logičke sheme*** spada prikaz znanja temeljen na ***matematičkoj logici***, kako standardnoj tako i nestandardnoj, dok se postupak zaključivanja temelji na pravilima zaključivanja. Možda najvažniji predstavnik logičkih shema su ***produkcijski sustavi*** temeljeni na logičkoj implikaciji[57]. Ovakav način pohrane znanja već smo upoznali kod problema dvaju vrčeva. Znanje o tome kako iz jednog stanja prostora rješenja zadatka prijeći u sljedeće stanje formalno je bilo zapisano nizom logičkih implikacija. One su na lijevoj strani imale uvjet za primjenu pravila, a na desnoj rezultat do kojeg se došlo primjenom pravila. Možemo kazati da su produkcijski sustavi danas sigurno jedan od najrasprostranjenijih načina pohrane znanja pa ćemo ih u nastavku posebno obrađivati. U logičke sheme za prikaz znanja svakako trebamo uključiti i ***opisnu (deskriptivnu) logiku*** uvedenu kao formalni jezik za predstavljanje znanja u potpunosti temeljen na logici, a s njom je usko vezan i OWL (engl. *Web Ontology Language*), jezik za formalni zapis ontologija, pa ćemo o opisnoj logici više kazati pri kraju ovog poglavlja.

U ***deklarativne ili mrežne sheme***, koje se ponekad zovu i ***asocijativne sheme***, znanje se pohranjuje u obliku objekata (koncepata) i njihove relacijske i asocijativne povezanosti. Grafički prikaz ovakvog načina pohrane znanja su mrežni prikazi (liste, grafovi) kojima se naglašava povezanost pojedinih elemenata znanja. Mrežnim shemama pohranjuje se statičko znanje. Najistaknutiji predstavnici su ***semantičke mreže*** i ***konceptualne ovisnosti***. Ovdje bismo mogli na neki način uključiti i ***ontologije***, razvijene u okviru semantičkog Weba. Iako su ontologije primarno razvijene za formalno definiranje kategorija, svojstava i relacija između koncepata, entiteta i podataka unutar određene domene u svrhu razmjene znanja između različitih agenata, prije svega računalnih, a ne za pohranu znanja i formiranje baza znanja, one ipak predstavljaju strukturni način formalnog prikaza znanja iz neke domene. Kod projektiranja inteligentnih sustava obično se najprije kreće od definiranja ontologije domene u kojoj će inteligentni sustav djelovati.

Proceduralne sheme služe i za prikaz proceduralnog, dinamičkog znanja. Proceduralan prikaz znanja ujedno je i stvarni plan kako znanje koristi u dinamičkom rješavanju nekog zadatka. Spomenut ćemo ***scenarij***, proceduralnu shemu koja je direktno preuzeta iz stvarnog života i korištena kod pohrane znanja o dinamičkim postupcima. U proceduralne sheme neki autori ubrajaju i produkcijske sustave s obzirom na to da produkcijska pravila daju posljedicu primjene određenog pravila, ako je uzročni dio zadovoljen.

Okviri u svemu tome imaju posebno mjesto kao neka vrsta fuzije deklarativnih i proceduralnih metoda. Okviri su strukture preko kojih na primjer možemo implementirati semantičku mrežu kojoj smo dodali i proceduralni dodatak kako manipulirati pohranjenim znanjem.

U nastavku se kratko opisuju osnovne značajke najznačajnijih struktura za pohranu znanja, a tko želi naučiti više, upućujemo ga na literaturu u posljednjem poglavlju.

4.4.1 Produkcijski sustavi

Logički sustavi služe prije svega za osnovno prikazivanje (kodiranje) znanja, a postoji niz primjera vrlo uspješnih inteligentnih sustava kod kojih je znanje i strukturno pohranjeno samo u obliku logičkih propozicija. Tipičan primjer su ***produkcijski sustavi*** (engl. *Production Systems),* ponekad zvani i ***produkcijski sustavi pravila*** (engl. *Production Rule Systems*) ili ***sustavi temeljeni na pravilima*** (engl. *RDSs – Rule Based Systems*), kod kojih su osnovni dijelovi baze znanja uzročno-posljedična ***„ako… onda…"*** pravila (engl. *If … Then … Rules*), matematički formalizirana logičkom implikacijom. Ovakve baze znanja imaju posebno mjesto u sustavima umjetne inteligencije, a s njima smo se već susreli kod problema dvaju vrčeva.

[57] Napomenimo da neki autori produkcijske sustave uključuju u proceduralne sheme za prikaz znanja, međutim kako se u potpunosti temelje na formalnoj logici i logičkim pravilima zaključivanja, u okviru ovog teksta ipak smo ih uključili u logičke sheme za pohranu znanja.

Rečenicu „*Napuni 4-litarski vrč.*" (iz problema dvaju vrčeva) formalno zapisujemo logičkom implikacijom $(x, y \mid x < 4) \rightarrow (4, y)$, čija bi jezična interpretacija bila: ***„Ako** vrč od 4 litre nije pun, **onda** ga napuni do vrha."* Na lijevoj je strani formalni opis situacije kod kojeg možemo primijeniti ovo pravilo, a na desnoj posljedica, stanje koje ćemo dobiti nakon što pravilo primijenimo. Lijeva je strana ***„Ako..."*** uvjet, stanje, premisa, ***antecedenta***, a desna ***„onda..."*** posljedica, akcija, zaključak, ***konsekventa***.

Matematičar i logičar ***Emil Leon Posta*** (1987. – 1954.) jedan je od prvih znanstvenika koji se bavio produkcijskim pravilima, pa je između ostalog dokazao da se svaki matematički ili logički sustav može prikazati nekom vrstom produkcijskih pravila. U području umjetne inteligencije početkom 1970-ih godina uvode ih pioniri umjetne inteligencije ***Allen Newell*** i ***Herbert A. Simon*** nazivajući svako produkcijsko pravilo ***„osnovnom granulom znanja***" (engl. *Chunk of Knowledge*).

Formalni zapis jednog produkcijskog pravila je logička implikacija $P \rightarrow Q$, a produkcijski sustav sastoji se od n produkcijskih pravila međusobno povezanih logičkim operatorom disjunkcije ∨ (veznik ili):

$$(P_1 \rightarrow Q_1) \vee (P_2 \rightarrow Q_2) \vee \ldots \vee (P_n \rightarrow Q_n) \qquad (4\text{-}17)$$

Na primjer, kod jednostavnog opisa načina funkcioniranja sustava grijanja s tri pravila (koji ćemo spominjati kod neizrazitog zaključivanja) možemo pisati:

(1) ***Ako** je temperatura niska, **onda** grijanje jako. ili*

(2) ***Ako** je temperatura srednja, **onda** grijanje srednje. ili*

(3) ***Ako** je temperatura visoka, **onda** grijanje malo.*

I na lijevoj strani antecedente i na desnoj strani konsekvente može biti i složeni logički izraz, na primjer kod sustava koji upravlja i temperaturom i vlažnošću zraka produkcijsko pravilo može biti:

***„Ako** je temperatura niska i vlažnost niska, **onda** grijanje jako i ovlaživanje jako."*

što formalno možemo zapisati:

$$(P_1 \wedge R_1) \rightarrow (Q_1 \wedge U_1)$$

Na primjer, u dijagnostičkom sustavu pokretanja automobila možemo definirati tri pravila:

***„Ako** automobil neće upaliti, **onda** provjerite bateriju i provjerite gorivo."*

***„Ako** baterija slaba, **onda** napunite bateriju."*

***„Ako** goriva nema, **onda** napunite gorivo."*

što formalno zapisujemo:

$$P_1 \rightarrow (Q_1 \wedge U_1)$$

$$Q_1 \rightarrow S_1$$

$$U_1 \rightarrow T_1$$

Temeljni postupak zaključivanja kod produkcijskog sustava je ***eliminacija implikacije*** i to ***modus ponens***, koji smo već spominjali kod silogističkog zaključivanja:

Ivan = muškarac

svi muškarci → dječaci

Ivan = dječak

i ***modus tollens***, ali o njima više u poglavlju o logičkom zaključivanju. Ovdje ćemo samo spomenuti ***ulančavanje*** (engl. *Chaining*) produkcijskih pravila. Ulančavanje može biti ***unaprijed*** (engl. *Forward Chaining*) ili ***unatrag*** (engl. *Backward Chaining*). Na primjer, kod primjera dijagnostičkog sustava pokretanja automobila ulančavanje unaprijed bilo bi:

***„Ako** automobil neće upaliti, **onda** provjerite bateriju."* → ***„Ako** je baterija slaba, **onda** napunite bateriju."*
a unatrag:

*„**Ako** se bateriju treba napuniti, **onda** znači da je slaba.”* → *„**Ako** je baterija slaba, **onda** automobil neće upaliti.”*

Poseban problem produkcijskih sustava je ***pojava konflikata***. Konflikt je situacija kada nekoliko pravila iz baze znanja zadovolje trenutni uvjet (stanje, premisu, antecedentu). Pitanje je koje pravilo primijeniti? Kako bi se razriješila ova situacija, produkcijski sustav treba imati ***mehanizam razrješavanja konflikata*** (engl. *Conflict Resolution Strategy*). To je moguće ostvariti na više načina:

- ***red pojavljivanja pravila*** – primijeni pravilo koje se prvo pojavilo
- ***vremenski zadnje pravilo*** – primijeni pravilo koje je po vremenu zadnje dodano
- ***do sada nekorišteno pravilo*** – primijeni pravilo koje do sada nije bilo korišteno
- ***pravilo koje je više određeno*** – primijeni pravilo koje najviše sužava ulazne uvjete (ima najviše ulaznih uvjeta)
- ***proizvoljno*** – primijeni pravilo slučajnim izborom.

Prolog je programski jezik u potpunosti prilagođen programiranju produkcijskih sustava. Primjer realizacije u Prologu[58] problema punjenja dvaju vrčeva je:

```
% Pravila pretakanja
move(s(X,Y),s(4,Y)):- X is 0.
move(s(X,Y),s(X,3)):- Y is 0.
move(s(X,Y),s(0,Y)):- X > 0.
move(s(X,Y),s(X,0)):- Y > 0.
move(s(X,Y),s(0,Z)):- Z is X + Y, Z =< 3.
move(s(X,Y),s(Z,0)):- Z is X + Y, Z =< 4.
move(s(X,Y),s(Z,3)):- Z is X - (3 - Y), Z >= 0.
move(s(X,Y),s(4,Z)):- Z is Y - (4 - X), Z >=0.

% Zaključivanje
moves(Xs):- moves([s(0,0)],Xs).
moves([s(X0,Y0)|T], [s(2,Y1),s(X0,Y0)|T])
   :- move(s(X0,Y0),s(2,Y1)), !.
moves([s(X0,Y0)|T],Xs):-
    move(s(X0,Y0),s(X1,Y1)),
    not(member(s(X1,Y1),[s(X0,Y0)|T])),
    moves([s(X1,Y1),s(X0,Y0)|T],Xs).
pokreni:- moves(Xs), write(Xs), nl, fail.
```

Početno stanje je *s(0,0)*, a konačno stanje *s(2,Y1)*[59]. Pokretanjem naredbe `pokreni` dobije se dio kraćih rješenja[60], a među njima su i dva najkraća od 7 koraka koja smo već naveli u Poglavlju 2 kod definiranja problema dvaju vrčeva.

[58] 58 Korišten je *online* Prolog interpreter - https://swish.swi-prolog.org

[59] Napomena: Ako želimo neko drugo završno stanje, npr. *(2,0)*, tada *s(2,Y1)* na oba mjesta trebamo zamijeniti sa *s(2,0)*, a ako želimo neko drugo početno stanje, tada *s(0,0)* trebamo zamijeniti s njim.

[60] Napomena: Ako želimo dobiti sva moguća rješenja, treba ukloniti ! iz naredbe:
`moves([s(X0,Y0)|T], [s(2,Y1),s(X0,Y0)|T] :- move(s(X0,Y0),s(2,Y1)), !.`

```
?- pokreni
[s(2,3),s(4,1),s(0,1),s(1,0),s(1,3),s(4,0),s(0,0)]
[s(2,0),s(0,2),s(4,2),s(3,3),s(3,0),s(0,3),s(1,3),s(4,0),s(0,0)]
[s(2,0),s(0,2),s(4,2),s(3,3),s(3,0),s(0,3),s(4,3),s(4,0),s(0,0)]
[s(2,3),s(4,1),s(0,1),s(1,0),s(1,3),s(4,0),s(4,2),s(3,3),s(3,0),s(0,3),s(0,0)]
[s(2,0),s(0,2),s(4,2),s(3,3),s(3,0),s(0,3),s(0,0)]
[s(2,3),s(4,1),s(0,1),s(1,0),s(1,3),s(4,0),s(4,3),s(0,3),s(0,0)]
false
```

Algoritam traženja je povratni (engl. *Backtracking*) pa je zato početno stanje krajnje desno, a završno na početku. Drugi programski jezik posebno razvijen za izgradnju produkcijskih sustava je ***Jess***[61]. Jess je pisan u Javi, a kod procesiranja pravila koristi ***Rete algoritam***[62], vrlo efikasan algoritam za rješavanje složenijih poklapanja.

Produkcijskim sustavima još ćemo se vraćati u poglavlju 4.4.6 gdje govorimo o ontologijama i zaključivanju korištenjem ontologija, te posebno u poglavlju 5 u kojem detaljno obrađujemo problematiku zaključivanja i rasuđivanja.

4.4.2 Semantičke mreže

Mrežne strukture za prikaz znanja imaju dugu povijest. Grčki neoplatonski filozof **Porphyry iz Thyre** (234. – 305.) predložio je hijerarhijski prikaz oblika stabla s korijenom na vrhu za prikaz Aristotelovih kategorija. Sličnu strukturu imaju i semantičke mreže koje se u umjetnoj inteligenciji koriste za prikaz statičkog, deklarativnog znanja, inspirirane prije svega pretpostavkom vezanom uz to kako funkcioniraju ljudske ***dugotrajne asocijativne memorije*** (engl. *Long-Term Associative Memories*). Pretpostavlja se da se ljudsko dugotrajno pamćenje temelji na relacijski povezanim različitim činjenicama, na primjer kada osjetimo miris jela koji je u djetinjstvu kuhala baka, miris povezujemo i s bakom i s jelom koje je ona kuhala.

U računarstvu se ideja semantičkih mreža prvi put pojavila ranih 1960-ih godina i bila je vezana uz problematiku strojnog prevođenja prirodnog jezika. Semantičke mreže prvi je spomenuo ***Richard Hook Richens*** (1919. – 1984.) 1956. godine kao međujezik za strojno prevođenje prirodnih jezika. Osnovna je ideja semantičkih mreža pohraniti znanje u obliku semantičkih relacijskih veza između koncepata. ***Margarete Masterman*** (1910. – 1986.) je 1961. godine definirala skup od 100 primitivnih tipova koncepata za definiranje 15.000 koncepata. U konačnom obliku definirala ih je grupa znanstvenika ranih 1960-ih godina radeći na projektu SYNTHEX, a ***M. Ross Quillian*** ih je u konačnoj formi uobličio u svojoj doktorskoj disertaciji *„Semantic Memory”* 1966. godine (Quilian, 1966., 1967.).

Postoje različiti tipova semantičkih mreža, a možda su najzanimljivije hibridne koje uključuju različite tipove, npr. ***definicijske mreže*** (engl. *Definitional Networks*) koje naglašavaju podtipove koncepata i koriste poveznice klasa i sub-klasa (relacija ***„isa”*** – detalji u nastavku) i ***izjavne mreže*** (engl. *Assertional Networs*) kod kojih se izjavljuju propozicije npr. svojstva koncepata (relacija ***„has”*** – detalji u nastavku).

Grafički prikaz semantičke mreže je usmjereni graf gdje su čvorovi koncepti, a usmjerene grane relacijska veza između njih. Kod semantičkih mreža i ljudskih asocijativnih mreža imamo koncepte u čvorovima i relacijske veze između njih, s tim da su kod semantičkih mreža relacijske veze formalizirane standardnim relacijama.

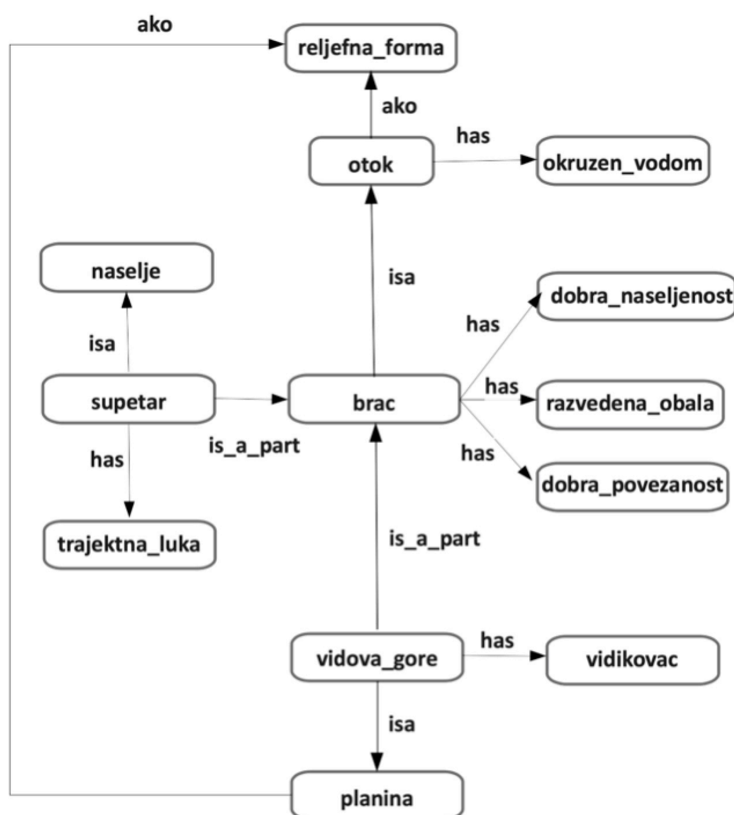
Slika 4-11 prikazuje primjer semantičke mreže kojom opisujemo otok Brač.

---

i zamijeniti je naredbom:
```
moves([s(X0,Y0)|T], [s(2,Y1),s(X0,Y0)|T] :- move(s(X0,Y0),s(2,Y1)).
```

[61] https://www.jessrules.com/jess/

[62] https://www.jessrules.com/doc/71/rete.html



Slika 4-11. Dio semantičke mreže kojom opisujemo otok Brač

Čvorovi su koncepti (pojmovi), a grane relacije između njih. Standardne relacije su:

- ***„isa"*** – od engleskih riječi *„is a"* – u hrvatskom prijevodu *„je"* povezuje klasu i super-klasu, npr. *„Brač je otok"* (u semantičkoj mreži formalno *„Brač* ***isa*** *otok"*) ili cijeli slijed nasljeđivanja: *„srdela* ***isa*** *plava_riba"* → *„plava_riba* ***isa*** *riba"* → *„riba* ***isa*** *životinja"* → *„životinja* ***isa*** *živo_biće"*.
- ***„ako"*** – od engleskih riječi *„a kind of"* ili ponekad ***„inst"*** ili *„instance of"* – u hrvatskom prijevodu *„je vrsta"* povezuje jedinku i klasu npr. *„otok je vrsta reljefnog oblika"* (u semantičkoj mreži formalno *„otok* ***ako*** *reljefni_oblik"*).
- ***„is_a_part"*** – u hrvatskom prijevodu *„je dio"* – povezuje svojstveni dio i cjelinu, npr. *„Supetar je dio Brača"* (u semantičkoj mreži formalno *Supetar* ***is_a_part*** *Brač*). Pogledajmo i primjer s nasljeđivanjem: *„leća* ***is_a_part*** *oka"* → *„oko* ***is_a_part*** *glave"* → *„glava* ***is_a_part*** *ribe"*.
- ***„has"*** – u hrvatskom prijevodu *„ima"* – povezuje objekt i njegove atribute, npr. *„Brač ima razvedenu obalu"* (u semantičkoj mreži formalno *„Brač* ***has*** *razvedenu_obalu"*). Ponekad se koristi i ***„prop"*** od engleske riječi *„property"* – u prijevodu svojstvo.
- ***relacije svojstava*** – koje se proizvoljno nazivaju ovisno o kojim svojstvenim konceptima govorimo. Kod otoka Brača to su *„naseljenost"*, *„povezanost"*, *„poljoprivredne_kulture"*, itd. Na primjer, u relaciji: *„Brač je dobro naseljen."* (u semantičkoj vezi formalno iskazano *„Brač* ***naseljenost*** *dobra"*). U primjeru na slici 4-10 nismo koristili proizvoljne relacije svojstava već smo sve relacije nastojali svesti na standardne, pa smo umjesto *„Brač* ***naseljenost*** *dobra"* koristili *„Brač* ***has*** *dobru_naseljenost"*.

Sliku 4-10 bismo mogli opisivati s pola stranice teksta: *„Brač je otok. Otoci su vrste reljefnog oblika. Dio otoka Brača je Supetar. Supetar je naselje. Supetar ima trajektnu luku. Brač je dobro naseljeni otok* ...". Semantička mreža sve to vrlo jednostavno prikazuje i što je možda još i važnije omogućava

zaključivanje koristeći prije svega principe ***nasljeđivanja*** ili ***tranzitivnosti*** (engl. *Inheritance*). Ako *x* ima neka svojstva, a *y* je na primjer relacijom *„isa*" vezan s njim, onda je i *y* preuzeo sva ta svojstva. Na primjer, uzmimo relaciju *„opis*" koja za otok kaže *„okružen vodom*" što bismo jezično iskazali *„Otok je sa svih strana okružen vodom*". Kako vrijedi *„Brač je otok*", onda je on i preuzeo sva svojstva otoka pa tako vrijedi i *„Brač je sa svih strana okružen vodom.*".

Standardne relacije u predikatnoj logici najčešće uzimamo kao predikate:

isa(Brač,otok) = isa(x,y)

isa(otok,reljefni_oblik) = isa(y,z)

pa tada tranzitivnost definiramo implikacijom:

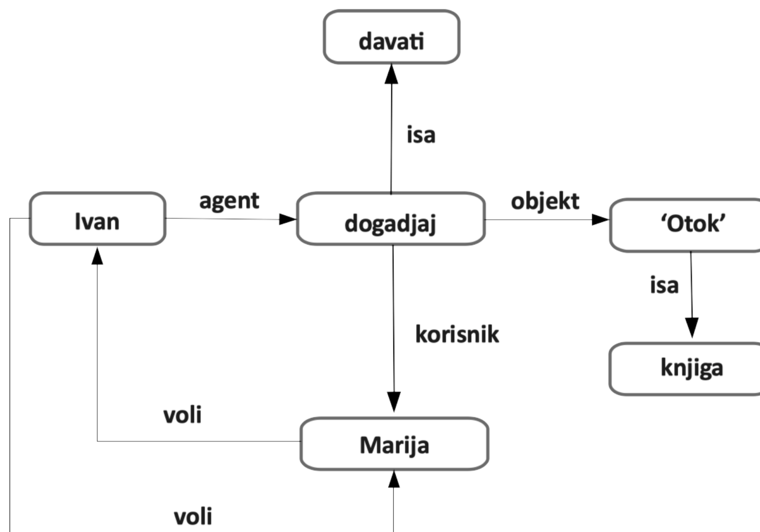
$\forall x \forall y \forall z \quad isa(x,y) \wedge isa(y,z) \rightarrow isa(x,z)$

što jezično interpretiramo kao *„Kako je Brač otok, a otok je reljefni oblik, onda je i Brač reljefni oblik".*

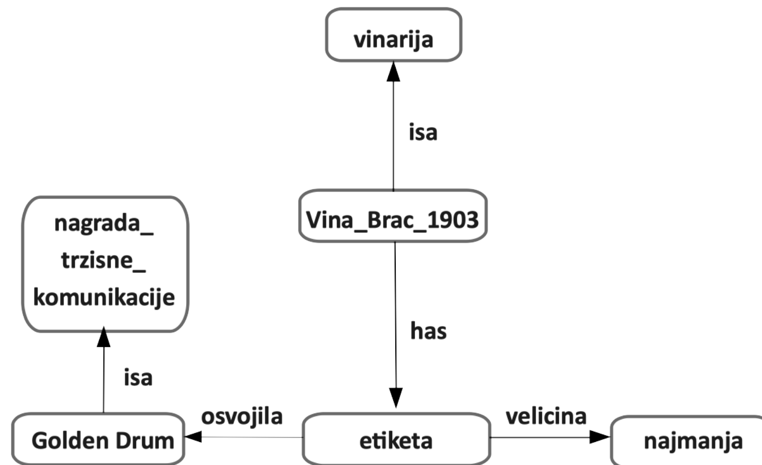
Zaključivanje u semantičkim vezama temelji se na praćenju veza u mreži. Nasljeđivanje postaje složenije u slučaju višestrukog nasljeđivanja kada koncept pripada većem broju kategorija (npr. da smo za otok kazali istovremeno i da je vrsta reljefnog oblika i da je vrsta konceptualnog iskazivanja pojma usamljenosti). Nasljeđivanje u tom slučaju nije jednoznačno, te vraća različite koncepte koji ponekad mogu biti i kontradiktorni. Vezano uz ovaj primjer naglasimo da u semantičkim mrežama postoje i disjunktivne (ili) i konjunktivne (i) standardne veze. Na primjer, za disjunktivnu vezu *„Otok je reljefni oblik" ili „Otok je konceptualni pojam usamljenosti*", a za konjunktivnu *„Supetar je dio Brača*" i *„Supetar je bračka općina*".

Na kraju pogledajmo još i kako možemo u obliku semantičkih mreža pohraniti znanje iskazano rečenicama prirodnog jezika korištenjem proizvoljnih relacijskih svojstava:

Ivan i Marija se vole i Ivan je dao Mariji knjigu „Otok".



Vina vinarije „Vina Brač 1903' imaju najmanju etiketu koja je dobila nagradu Golden Drum za tržišnu komunikaciju.



Slika 4-12. Dva jezična iskaza i odgovarajuće semantičke mreže

Semantičke mreže jednostavno se definiraju u standardnim programskim jezicima umjetne inteligencije Lispu i Prologu. Na primjer, koncept Brač sa slike 4-10 u Lispu možemo definirati:

```
(setq *database*
'((brac  (isa otok)
         (has dobra_naseljenost)
   (has razvedena_obala)
         (has dobra_povezanost))
  (supetar       (isa naselje)
          (is_a_part brac)
                 (has trajektna_luka))))
```

Algoritam 4-1 daje primjer realizacije cijele semantičke mreže sa slike 4-10 u Prologu, te primjere zaključivanja. Kod definiranja pravila nasljeđivanja korištena je rekurzija i proceduralna apstrakcija.

Algoritam 4-1. Realizacija semantičke mreže koja opisuje dio otoka Brača sa slike 4-10 u Prologu

```
% Činjenice
isa(brac,otok).
isa(supetar,naselje).
isa(vidova_gora,planina).
%
ako(otok,reljefna_forma).
ako(planina,reljefna_forma).
%
is_a_part(supetar,brac).
is_a_part(vidova_gora,brac).
%
has(okruzen_vodom,otok).
has(dobra_naseljenost,brac).
```

```
has(razvedena_obala,brac).
has(dobra_povezanost,brac).
has(vidikovac, vidova_gora).
has(trajektna_luka,supetar).
%
% Pravilo podklase
podklasa(X,Y):-isa(X,Y).
podklasa(X,Y):-isa(X,Z),podklasa(Z,Y).
%
% Pravilo primjerka
primjerak(X,Y):-ako(X,Y).
primjerak(X,Y):-ako(X,Z),podklasa(Z,Y).
%
% Pravilo biti_dio
biti_dio(X,Y):-is_a_part(X,Y).
biti_dio(X,Y):-is_a_part(X,Z),biti_dio(Z,Y).
%
% Pravilo svojstva
svojstvo(X,Y):-has(X,Y).
svojstvo(X,Y):-primjerak(Y,Z),svojstvo(X,Z).
svojstvo(X,Y):-podklasa(Y,Z),svojstvo(X,Z).
svojstvo(X,Y):-biti_dio(Y,Z),svojstvo(X,Z).
```

Primjer zaključivanja[63] s ovom semantičkom mrežom:

```
? - podklasa(brac,Y)
Y=otok
?- primjerak(X,Y)
X=otok
Y=reljefna_forma
X=planina
Y=reljefna_forma
? - biti_dio(X,brac)
X=supetar
X=vidova_gora
? - svojstvo(X,brac)
X=dobra_naseljenost
X=razvedena_obala
X=dobra_povezanost
X=okruzen_vodom
```

[63] Korišten je *online* Prolog interpreter - https://swish.swi-prolog.org

Semantičke mreže imaju brojne nedostatke. Neki od njih su sljedeći: relacije nisu u potpunosti standardno definirane, ne razlikuju se klase i individualne jedinke, relacije između čvorova su samo binarne relacije (pa je nemoguće opisivati predikatne izjave od 3 ili više argumenata). Zbog ovih nedostataka semantičke mreže i nisu pronašle baš široku primjenu u praktičnim primjenama umjetne inteligencije, ali su vrlo važne za shvaćanje koncepata pohrane znanja.

4.4.3 Scenarij

Scenarij (engl. *Script*) je strukturni prikaz koji opisuje stereotipne sekvence događanja u određenom kontekstu. Scenarijom organiziramo informacije zajedničke za uobičajene događaje (npr. odlazak u restoran, postupak kupnje, način vođenja procesa, postupanje u kriznim situacijama itd.). Pretpostavlja se da na sličan način i ljudi pamte tipične procedure. Scenarij je formalno predložen 1977. godine u radu ***Rogera Carla Schanka*** i ***Roberta Abelsona*** (Schank, Abelson, 1977) s ciljem organiziranja struktura konceptualne ovisnosti u opisima tipičnih situacija. Teoriju ***konceptualne ovisnosti*** (engl. *Conceptual Dependency*) nekoliko su godina prije (1974. godine) predložili ***Roger Carl Schank*** i ***Charles J. Riegel*** (Schank, Riegel, 1974) za formalno modeliranje dubokih semantičkih struktura prirodnog jezika.

Svaki scenarij ima sljedeće komponente:

- ***Ulazne uvjete*** (engl. *Entry Conditions*) – uvjeti koji moraju biti ispunjeni prije nego li se scenarij izvede.
- ***Rezultat*** (engl. *Result*) – situacija koju ćemo dobiti na kraju izvedbe scenarija.
- ***Rekvizite*** (engl. *Props*) – objekti koji su opisani u scenariju.
- ***Uloge*** (engl. *Roles*) – osobe koje sudjeluju u scenariju.
- ***Varijacije*** (engl. *Track*) – varijacija univerzalnog ponašanja koju opisuje scenarij. Varijacije istog scenarija dijele neke zajedničke osobine, ali ne sve.
- ***Scene*** (engl. *Scenes*) – sekvenca događaja.

Slika 4-12 prikazuje jedan tipičan scenarij postupka konzumacije hrane u standardnom restoranu. Svaku scenu možemo direktno opisati njezinim slijedom događaja ili pozvati njezin scenarij, pa npr. umjesto definiranja scene *„ulazak"* pomoću predikatne logike kako pokazuje slika 4-13 može se koristiti i poseban scenarij *„ulazak"* koji će imati iste rekvizite i uloge. Ovaj scenarij vrijedi za standardne restorane kod kojih konobar prima narudžbu, dok bi za neki drugi tip restorana scenarij bio potpuno različit.

SCENARIJ	restoran
VARIJACIJA	kafić
REKVIZITI	restoran, stol, jelovnik, hrana, novac
ULOGE	gost, konobar, kuhar
ULAZNI UVJETI	gladan(gost) & ima_novac(gost)
REZULTAT	¬ gladan(gost) & ima_manje_novca(gost) & ima_više_novca(konobar)
SCENA 1	**ULAZAK** ulazi(gost, restoran). bira(gost, stol). sjeda(gost, stol).

SCENA 2	**NARUČIVANJE** uzima(gost, jelovnik). bira(gost, hrana). zove(gost, konobar). naručuje(gost, hrana). bilježi(konobar, hrana). izlazi(konobar, restoran). kuha(kuhar, hrana).
SCENA 3	**KONZUMIRANJE** daje(kuhar, hrana, konobar). ulazi(konobar, restoran). nosi(konobar, hrana). jede(gost, hrana).
SCENA 4	**PLAĆANJE** vadi(gost, novac). uzima(konobar, novac).
SCENA 5	**IZLAZAK** diže_se(gost, stol). izlazi(gost, restoran).

Slika 4-13. Scenarij koji opisuje događanja u standardnom restoranu – adaptirano prema primjeru (Schank and Abelson, 1977)

U ovom primjeru sva događanja u restoranu su kodirana predikatnom logikom, s tim da su predikati akcije (događanja), a uloge i rekviziti su predikatni argumenti (konstante – elementi domene). Da bi se scenarij primijenio, ulazni uvjet `(gladan(gost) & ima_novac(gost))=1` treba biti istinit, a po završetku scenarija istiniti su rezultati `(sit(gost) & ima_manje_novca(gost) & ima_više_novca(konobar)) = 1`. Primjer je pojednostavljen u odnosu na originalni pristup utemeljen na ***konceptualnim ovisnostima*** koji su ***Schank i Abelson*** koristili u svom radu. Konceptualne ovisnosti formaliziraju tipične akcije, a neke od njih potrebne za scenarij *„restoran"*[64]:

`ATRANS` – promjena nosioca veze (glagol dati)

`PTRANS` – prijenos fizičke lokacije objekta (glagol ići)

`ATTEND` – fokusirati osjetilni organ (glagol zapaziti)

`MBUILD` – mentalno stvarati nove informacije (odlučiti)

`MOVE` – premjestiti tijelo ili dio tijela od strane vlasnika (glagol pokrenuti)

`INGEST` – progutati objekt od stane živog bića (glagol jesti).

Konceptualne ovisnosti uvode i pojam **agenta** i **objekta**. Agent provodi akciju nad objektom, a objekt može biti i on sam.

[64] Za više detalja pogledati radove (Schank i Rieger 1974) i (Schank i Abelson 1977).

Korištenje ovih oznaka Scena 1 – ULAZAK može se formalno zapisati na sljedeći način:

```
gost PTRANS u restoran
gost ATTEND oči na stolove
gost MBUILD stol
gost PTRANS gost do stol
gost MOVE gost u poziciju sjedenje
```

Scena 3 – KONZUMIRANJE može se formalno zapisati na sljedeći način:

```
kuhar ATRANS hrana konobar
konobar PTRANS restoran
konobar ATRANS hrana gost
gost INGEST hrana
```

Scenu 1 možemo još formalnije zapisati u duhu Lisp programskog jezika koristeći pojmove agent i objekt:

```
PTRANS(agent(gost))(objekt(gost))(to(restoran))
(ATTEND(agent(gost))(objekt(oči))(to(stol))
(MBUILD(agent(gost))(objekt(gost))(to(stol))
(PTRANS(agent(gost))(objekt(gost))(to(stol))
(MOVE(agent(gost))(objekt(gost))(position(sjedenje)))
```

Scenarij, kao i semantičke mreže, ima brojna ograničenja, ali je ipak pronašao praktičnu primjenu kod ograničenog razumijevanja tekstualnih priča, jednog od zahtjevnijih zadataka u području umjetne inteligencije. Jedan od primjera je SAM (engl. *Script Applier Mechanism*) kojeg je 1978. godine predložio ***Richard Edward Cullingford***. SAM je omogućavao razumijevanje jednostavnih novinskih tekstova uz odgovaranje na pitanja vezana uz sadržaj teksta.

4.4.4 Okviri

Strukturno prikazivanje znanja okvirima nastalo je kao rezultat proučavanja načina na koje ljudi analiziraju novu situaciju. Postoje brojni eksperimenti koji navode na misao da ljudi informacije koje pamte organiziraju u strukture po zajedničkim karakteristikama i svojstvima koje nadopunjavaju novim iskustvom, činjenicom ili zaključkom. Polovinom 1970-ih godina pojavila se ideja da se i znanje strukturno organizira u slične strukture nazvane ***okviri*** (engl. *Frames*). Teorija okvira inicijalno je započela s MIT tehničkim izvještajem ***Marvina Minskyja*** *„Okvir za predstavljanje znanja*" (engl. *A Framework for Representing Knowledge*) iz 1974. godine. Minsky je predložio shemu za pohranu znanja kod koje bi se znanje pohranjivalo u paketima nazvanim okviri koji su bili uključeni u mrežu nazvanu ***sustav okvira*** (engl. *Frame System*) u kojoj pojedini okvir ukazuje na druge za njega relevantne okvire obično koristeći standardne relacije semantičkih mreža (*a_kinf_of, is_a_part, is_a*, itd.).

Ideja je krenula od toga da npr. izjava *„Brač je otok.*" zapravo znači *„Brač je jedinka koja pripada klasi otok.*", ali u konačnoj varijanti okviri su definirani kao složene semantičke mreže organizirane tako da se pomoću njih mogu prikazati zajedničke osobine stvari. Okvir je struktura koja sadrži podatke ili procedure koje se odnose na isti objekt, organizirane tako da se cijela struktura može uključivati ili isključivati u druge strukture. Okvir tipično opisuje određenu klasu objekata, kao na primjer stolicu ili sobu, pa ga nazivamo ***generički okvir*** (engl. *Generic Frame*). Okvir se može odnositi i na pojedini konkretni objekt, pa ga u tom slučaju nazivamo ***pojedinačni okvir*** (engl. *Instance Frame*). Svi pojedinačni okviri nasljeđuju sva svojstva svojih generičkih okvira. Na primjer, obavezno vrijednost okvira *„stolica"* treba biti *„sjedeća ploha"*. Stolica može biti s ili bez naslona, s tri ili četiri noge, međutim ako nema sjedeće plohe, onda objekt ne može biti stolica. U pojedinačnom okviru koji se odnosi na neku konkretnu stolicu ovo se svojstvo ne treba posebno upisivati, zato što se prenosi iz generičkog okvira stolica.

Okvir se sastoji se od tzv. ***SLOTS*** (*razdjeljaka*) kojima definiramo i opisujemo određene aspekte objekta. Razdjeljci imaju svoje dijelove koji se zovu ***FACETS*** (*stranice*). ***FACET*** može biti alfa-numerička vrijednost (***VALUE***), drugi okvir, ali može i pripadati različitim dodacima (***DEMONS***) koji definiraju procedure (akcije) koje treba poduzeti ako se pristupi tom ***SLOT***-u. Tipični DEMONS su:

- ***if_added*** – što treba napraviti kada se ***SLOT*** puni podacima (ovaj ***DEAMON*** obično prati i ***DEAMON range*** koji daje dopuštene vrijednosti novih podataka)
- ***if_removed*** – što treba napraviti kada se iz ***SLOT***-a uklone podaci
- ***if_changed*** – što treba napraviti kada se u ***SLOT***-u promijene podaci
- ***if_needed*** – što treba napraviti kada se zatraže podaci iz ***SLOT***-a
- ***if_new*** – što treba napraviti kada se formira novi slot, a posebna mjesta imaju i
- ***help*** – kada se ***SLOT*** dohvati, a ne može se izvršiti zatraženi postupak (na primjer dodaje se vrijednost, a ona se ne uklapa u ***range***), te
- ***default*** – vrijednost iz ***ranga*** koja se vraća, ako u ***SLOT****-u* nema konkretnih podataka, a DEAMON ***if_needed*** (ako postoji) ne može pronaći odgovor.

Kako bi se uočila razlika između strukture *„scenarij"* i *„okvir"*, ilustrativni primjer okvira *restoran* prikazuje slika 4-14. Okvir je povezan relacijskom vezom ***a_kind_of*** s generičkim okvirom poslovna ustanova. Postoji nekoliko ***SLOTS*** koji imaju svoj ***range, default*** vrijednost i proceduralni dodatak ***if_needed***. ***SLOT*** *vrijeme_rada* u svom *range* pokazuje na drugi generički okvir *radno_vrijeme*.

Okviri su najčešće međusobno povezani u semantičku mrežu. Primjer dijela sustava okvira koji prikazuju ustroj FESB-a prikazuje slika 4-15.

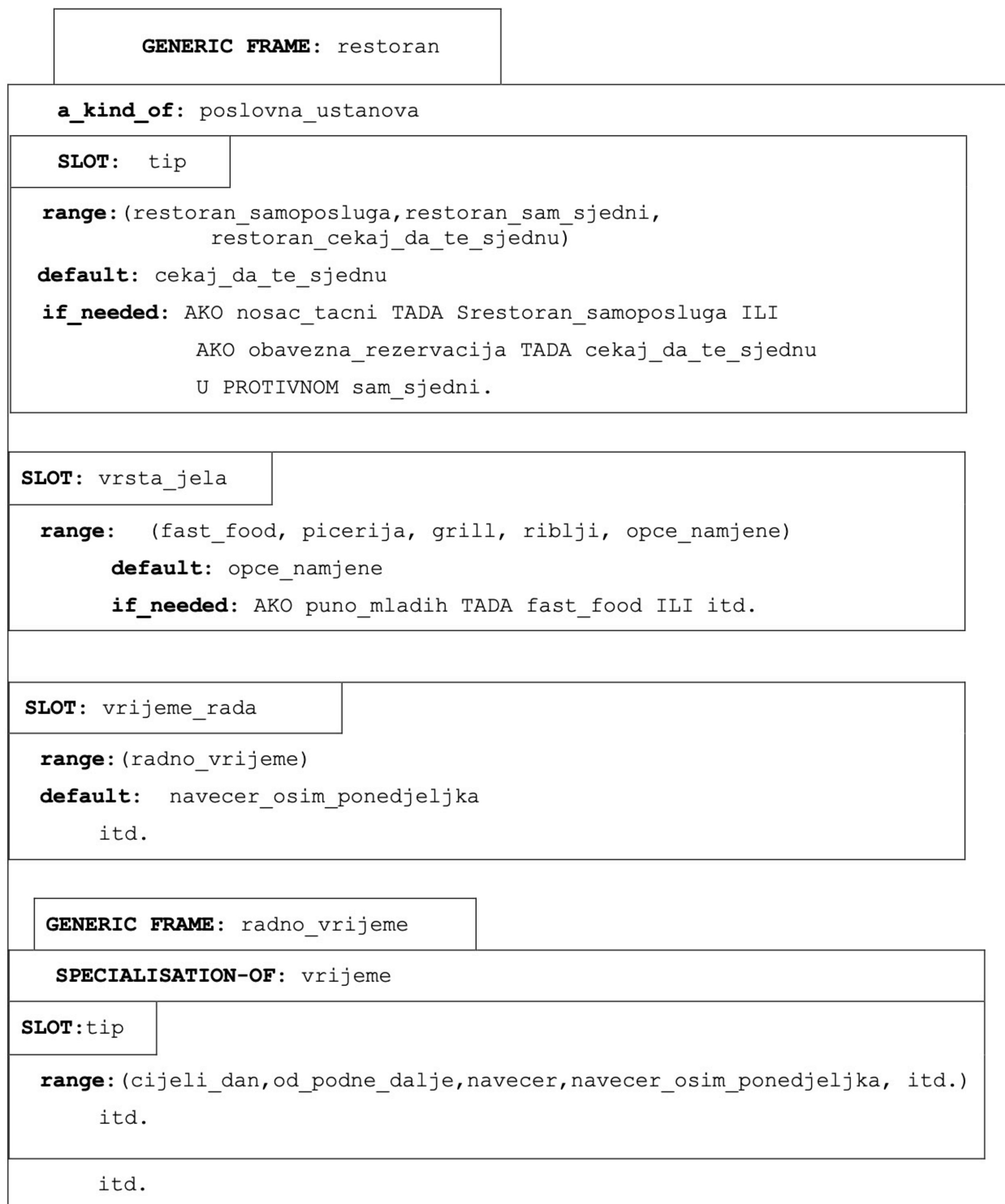
Na vrhu je generički okvir *„FESB",* a ispod su generički okviri podklase ustrojnih jedinica *„Zavod"* i *„Računski centar".* Na nižim razinama su pojedinačni okviri pojedinih *„Katedri"* i nekoliko nastavnika i suradnika pojedinih ustrojbenih jedinica. Generički okviri uobičajeno imaju ***default*** vrijednosti koje se mogu redefinirati u donjim slojevima. Da je umjesto *„default"* vrijednosti stavljena fiksna vrijednost, donji slojevi ne bi je mogli mijenjati, već bi je direktno preuzeli. Fiksne vrijednosti ostaju nepromijenjene za generičke okvire podklase i pojedinačne okvire.

Generički okviri uobičajeno imaju ***default*** vrijednosti koje se mogu redefinirati u donjim slojevima. Da je umjesto *„default"* vrijednosti stavljena fiksna vrijednost, donji slojevi ne bi je mogli mijenjati, već bi je direktno preuzeli. Fiksne vrijednosti ostaju nepromijenjene za generičke okvire podklase i pojedinačne okvire. Na primjer, u *frame_111* koji predstavlja katedru KaMIS namjerno smo napravili pogrešku i još jedanput unijeli ***SLOT*** predstojnik, sada s drugim imenom, međutim on ne može zamijeniti fiksnu vrijednost ***SLOT***-a predstojnik u podklasi Zavod za elektroniku i računarstvo kojoj KaMIS pripada. Isto tako svi okviri nasljeđuju ***SLOT***-ove koji u njima nisu posebno definirani. Na primjer, trebamo li telefon Zavoda za elektroniku i računarstvo, sustav okvira vratit će telefon FESB-a, koji je u hijerarhiji iznad jedino definiran.

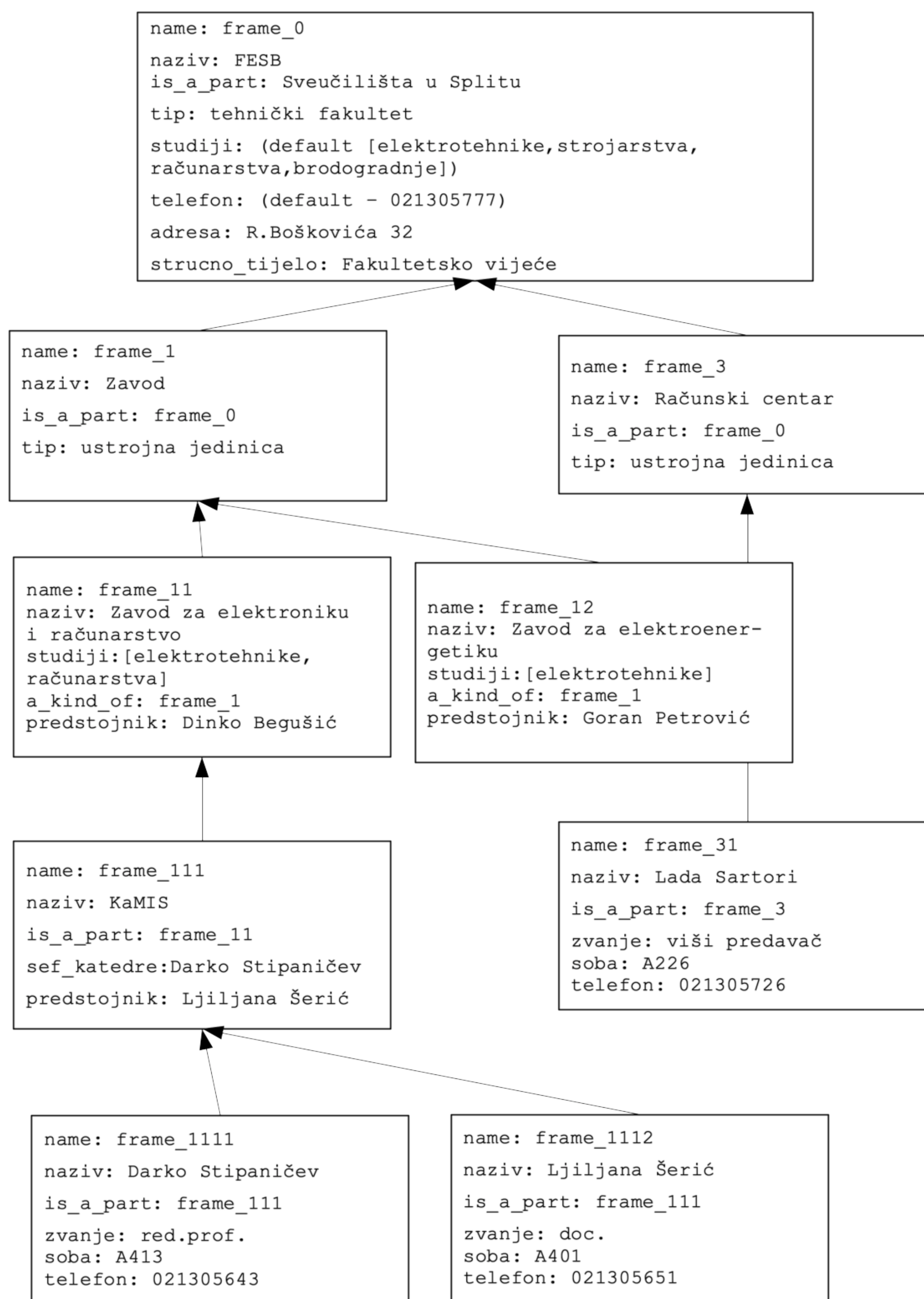
Na primjer, u *frame_111* koji predstavlja katedru KaMIS namjerno smo napravili pogrešku i još jedanput unijeli ***SLOT*** predstojnik, sada s drugim imenom, međutim on ne može zamijeniti fiksnu vrijednost ***SLOT***-a predstojnik u podklasi Zavod za elektroniku i računarstvo kojoj KaMIS pripada. Isto tako svi okviri nasljeđuju ***SLOT***-ove koji u njima nisu posebno definirani. Na primjer, trebamo li telefon Zavoda za elektroniku i računarstvo, sustav okvira vratit će telefon FESB-a, koji je u hijerarhiji iznad jedino definiran.

U ovom primjeru nemamo proceduralne dodatke. Mogli smo ih primjerice dodati za proračun godina pojedinog djelatnika, ako bi u njegovom okviru postojao ***SLOT*** `godina_rodjenja`, a poznata nam je i vrijednost varijable `sadasnja_godina`. U tom slučaju za ***SLOT*** *starost* stavili bismo proceduru:

```
starost: if_needed sadasnja_godina-godina_rodjenja
```



Slika 4-14. Dio generičkog okvira koji opisuje „restoran“



Slika 4-15. Dio generičkog okvira koji prikazuje ustroj FESB-a

Okviri su općenite strukture, pogodne za strukturno prikazivanje bilo kojeg znanja, a veliku primjenu pronašli su u sistemima dijagnostike pod kojima je nužno koristiti i određeno heurističko a i determinističko znanje. Prikaz znanja okvirima 1970-ih i 1980-ih godina bio je popriličnopopularan, pa su razvijeni i posebni jezici za prikazivanje i baratanje okvirima. Primjeri su *FRL – Frame Representation Language* (1977), *KRL – Knowledge Representation Language* (1977), *KL-ONE – Knowledge Language*

One (1980), *LOOPS – Lisp Object-Oriented Prigramming System* (1983) itd., ali se okviri jednostavno prikazuju i u standardnim jezicima umjetne inteligencije.

Kod pohrane znanja okvirima ***zaključivanje*** je slično kao kod semantičkih mreža i uglavnom se temelji na principima ***nasljeđivanja (tranzitivnosti)***. Na slici 4-14 i algoritmu 4-2 prikazan je primjer definiranja dijela generičkog okvira i zaključivanja po principu nasljeđivanja, a za realizaciju je korišten jezik Lisp.

Algoritam 4-2. Realizacija okvira dijela ustroja FESB-a sa Slike 4-14 u Lispu

```
;gnu clisp 2.49
; FESB
(setf (get 'FESB 'isa) '(Tehnički fakultet))
(setf (get 'FESB 'partOf) '(Sveučiliste u Splitu))
(setf (get 'FESB 'ime) '(FESB))
(setf (get 'FESB 'adresa) '(Ruđera Boškovića 32))
(setf (get 'FESB '(strucno_tijelo)) '(Fakultetsko vijeće))
(setf (get 'FESB 'telefon) '(305777))
(setf (get 'FESB 'studiji) (list 'Elektrotehnika 'Računarstvo 'Strojarstvo
'Brodogradnja '(Industrijsko inženjerstvo)))

; Zavod
(setf (get 'Zavod 'partOf) 'FESB)
(setf (get 'Zavod 'ime) '(Zavod))
(setf (get 'Zavod 'tip) '(Ustrojna jedinica))

; Računski centar
(setf (get 'Računski_centar 'partOf) 'FESB)
(setf (get 'Računski_centar 'ime) '(Računski centar))
(setf (get 'Računski_centar 'tip) '(Ustrojna jedinica))

; Zavod za elektroenergetiku
(setf (get 'Zavod_za_elektroenergetiku 'kindOf) 'Zavod)
(setf (get 'Zavod_za_elektroenergetiku 'ime) '(Zavod za elektroenergetiku))
(setf (get 'Zavod_za_elektroenergetiku 'predstojnik) '(Goran Petrović))
(setf (get 'Zavod_za_elektroenergetiku 'studiji) '(Elektrotehnika))

; Zavod za elektroniku i racunarstvo
(setf (get 'Zavod_za_elektroniku_i_računarstvo 'kindOf) 'Zavod)
(setf (get 'Zavod_za_elektroniku_i_računarstvo 'ime) '(Zavod za elektroniku i
računarstvo))
(setf (get 'Zavod_za_elektroniku_i_računarstvo 'predstojnik) '(Dinko Begušić))
(setf (get 'Zavod_za_elektroniku_i_računarstvo 'studiji) (list 'Elektrotehnika
'Računarstvo))

; Darko Stipaničev
```

```
(setf (get 'Darko_Stipaničev 'partOf) '(Zavod za elektroniku i racunarstvo))
(setf (get 'Darko_Stipaničev 'ime) '(Darko Stipaničev))
(setf (get 'Darko_Stipaničev 'zvanje) '(Redoviti profesor))
(setf (get 'Darko_Stipaničev 'ured) '(A413))
(setf (get 'Darko_Stipaničev 'telefon) '(305643))

; Ljiljana Šerić
(setf (get 'Ljiljana_Šerić 'partOf) '(Zavod za elektroniku i računarstvo))
(setf (get 'Ljiljana_Šerić 'ime) '(Ljiljana Šerić))
(setf (get 'Ljiljana_Šerić 'zvanje) '(Izvanredni profesor))
(setf (get 'Ljiljana_Šerić 'ured) '(A401))
(setf (get 'Ljiljana_Šerić 'telefon) '(305651))

; Lada Sartori
(setf (get 'Lada_Sartori 'partOf) '(Računski centar))
(setf (get 'Lada_Sartori 'ime) '(Lada_Sartori))
(setf (get 'Lada_Sartori 'zvanje) '(Viši predavač))
(setf (get 'Lada_Sartori 'ured) '(A226))
(setf (get 'Lada_Sartori 'telefon) '(305726))

; funkcije
(defvar *trazeno-svojstvo* '(trazeno_svojstvo))
(defun inherit-get (object property) (setq a (or (get object property)
(get-from-parents (or (get object 'partOf) (get object 'kindOf)) property))) (setf
*trazeno-svojstvo* a) )
(defun get-from-parents (parents property) (cond ((null parents) nil)
((atom parents) (inherit-get parents property)) (t (or (get-from-parents (car parents)
property)
(get-from-parents (cdr parents) property)))))
```

Primjer zaključivanja[65] s ovim sustavom okvira, kada se ne pronalazi podatak u okviru pa ga se traži u okvirima s kojima je traženi okvir povezan, je u nastavku:

```
(inherit-get 'Zavod_za_elektroniku_i_računarstvo 'telefon) (print *trazeno-svojstvo*)
(305777)
(inherit-get 'Darko_Stipaničev 'adresa) (print *trazeno-svojstvo*)
(RUĐERA BOŠKOVIĆA 32)
```

Dobra strana prikaza znanja okvirima je u tome što okviri grupiraju relacijski povezano znanje i što se jednostavno proširuju dodavanjem novih ***SLOTS***, a osnovni nedostatak je u tome što u sustavu okvira nije baš jednostavno provoditi zaključivanje.

[65] Korišten je *online* Lisp kompajler - https://rextester.com/l/common_lisp_online_compiler

4.4.5 Opisne logike

Opisne ili deskriptivne logike (engl. *DL – Descripton Logic*) uvedene su 1980-ih godina kao formalna struktura za predstavljanje znanja u potpunosti temeljena na logici, prije svega kako bi se prevladali nedostaci semantičkih mreža i okvira koji nisu imali formalnu logičku semantiku. DL-ovi su formalni jezici za predstavljanje znanja, teorijska osnova za implementaciju specijaliziranih jezika za prikaz i pohranu znanja, od kojih je najpoznatiji mrežni jezik OWL, najčešće korišten za definiranje ontologija (vidi poglavlje 4.4.6). DL-ovi se temelje na logici prvog reda (engl. *FOL – First Order Logic*), s tim da su terminologija i oznake malo drugačije (tablica 4-6).

Tablica 4-6. Terminologija predikatne logike prvog reda (FOL) i opisne logike (DL)

FOL	DL
konstanta	pojedinac
unarni predikat	koncept
binarni predikat	uloga

Postoji puno vrsta opisnih logika koje se dijele ovisno o tome koji su dozvoljeni operatori. Jedna od najjednostavnijih opisnih logika je $\mathcal{ALC}$ (engl. ***A****tributtive concept* ***L****anguage with* ***C****omplements*) koja znači da se radi o „*jeziku atributivnih koncepata s komplementom”*. Ova opisna logika modelira pojedince, koncepte i uloge, ali i relacije između njih. ***Pojedinac*** (engl. *Individual*) je individualni objekt domene, ***koncept*** (engl. *Concept*) predstavlja skup pojedinaca, a ***uloga*** (engl. *Role*) predstavlja binarnu relaciju između pojedinaca ili između koncepata. Na primjer, pojedinci su `sime, marija, rita` (tri osobe Šime, Marija i Rita), koncepti su unarni predikati, npr. `Musko(sime), Zensko(marija), Zensko(rita)`, što znači da je „*osoba Šime muško, a osobe Marija i Rita ženske*”, a uloge binarni predikat `imaDijete(sime,rita), imaDijete(marija,rita)`, što možemo interpretirati „*`Šime ima dijete Ritu, Marija ima dijete Ritu`*”. Ovo je zapis u obliku predikatne logike prvog reda (FOL), a $\mathcal{ALC}$ DL ima i svoje posebne simbole i oznake za operatore koje prikazujemo u tablici 4-7. $\mathcal{ALC}$ opisna logika je u biti pojednostavljena verzija predikatne logike prvog reda na način da je ograničena samo na binarne predikate.

U duhu DL-a, pojedince, koncepte i uloge koje smo spomenuli prije tablice sada zapisujemo:

- ***pojedinci:*** `sime, marija, rita`
- ***klase i pridruživanje pojedinaca klasama:*** `sime:Musko, marija:Zensko, rita:Zensko`
- ***pridruživanje uloga pojedincima:*** `(sime,rita):imaDijete, (marija,rita):imaDijete`. Napomenimo i to da se pridruživanje uloga ponekad zapisuje i `sime imaDijete rita, marija imaDijete rita`.

Tablica 4-7. Oznake i operatori ALC opisne logike i njihova interpretacija u predikatnoj logici prvog reda (FOL)

SIMBOL	OPIS	DL ZAPIS	FOL ZAPIS
`a , b`	`pojedinci`	`a , b`	`a , b`
`C , D`	`koncepti`	`C , D`	`C(a) , D(b)`
`R , P`	`uloge`	`R`	`R(a,b)`

⊤	koncept koji sadrži sve pojedince - vršni koncept *(Top Concept)*	⊤	
⊥	prazni koncept - donji koncept *(Bottom Concept)*	⊥	
⊓	Konjunkcija (presjek)	C⊓D	C(a) ∧ D(a)
⊔	Disjunkcija (unija)	C⊔D	C(a) ∨ D(a)
¬	negacija	¬C	¬C(a)
∀	univerzalno ograničenje	∀R.C	∃b(R(a,b)→C(b))
∃	egzistencijalno ograničenje	∃R.C	∃b(R(a,b) ∧ C(b))
⊑	inkluzija (hijerarhija) koncepata (podskup)	C ⊑D	∀a(C(a)→D(a))
≡	ekvivalencija koncepata	C ≡D	∀a(C (a)↔D(a))
:	pridruživanje pojedinaca konceptima	a:C	C(a)
:	pridruživanje uloga pojedincima	(a,b):R	R(a,b)

U opisnoj logici složene koncepte definiramo ekvivalencijom ili inkluzijom koncepata. Na primjer, pojam Otac definiramo na sljedeći način:

`Otac ≡ Musko⊓∃imaDijete.Osoba`

`Otac ⊑ Musko`

U prvom slučaju definirali smo potpuni uvjet, što znači da su svi pojedinci koji pripadaju konceptu `Otac` su nužno i pojedinci presjeka koncepta `Musko` i egzistencijalnog ograničenja `∃imaDijete` koncepta `Osoba`. Naravno, vrijedi i obrnuto, pa je `C ≡D` u biti kraći način zapisa `C ⊑D` i `D ⊑C`.

U drugom slučaju definiramo samo nužni uvjet za koncept `Otac`. Svi koji pripadaju konceptu `Otac` su podskup koncepta presjeka koncepata `Musko`. To znači ako je neki pojedinac član koncepta `Otac` onda je on nužno i član koncepta `LjudskoBice ⊓ Musko`, ali obrnuto ne vrijedi.

Koncepti vezani operatorima inkluzije (⊑) i ekvivalencije (≡) nazivaju se ***opće inkluzije koncepata*** (engl. *GCI – General Concept Inclusion*) ili ***terminološki aksiomi*** (engl. *Terminology Axioms*). Skup terminoloških aksioma označava se oznakom ***TBox*** čini prvi dio strukture ***baze znanja*** (engl. *KB – Knowledge Base*). Drugi dio strukture baze znanja je ***ABox*** skup ***aksioma tvrdnji*** (engl. *Assertional Axioms*) koji sadrži tvrdnje o konkretnim pojedincima i relacijama, na primjer:

`sime:Musko,marija:Zensko,rita:Zensko,(sime,rita):imaDijete,(marija,rita):imaDijete`

Baza znanja KB je uređeni par ***KB = (TBox , ABox)*** gdje ***TBox*** sadrži općenito znanje o domeni kojom se bavimo, a ***ABox*** informacije o konkretnim entitetima i odnosima unutar domene.

Važan dio opisne logike je zaključivanje i rasuđivanje. O samim pojmovima zaključivanja i rasuđivanja više u poglavlju 5, a ovdje samo spomenimo da se u opisnoj logici kod zaključivanja obično postavljaju pitanja vezana uz pripadanje članova ABox-a pojedinim konceptima, inkluzijama koncepata

i konzistentnosti TBox-a, a osnovni suvremeni algoritam rasuđivanja je tzv. ***„tableau" algoritam*** čiji detalji prelaze okvire ovog udžbenika[66], a ovdje ćemo samo prikazati kratki primjer. Tableau algoritam reducira problem rasuđivanja na problem zadovoljenja koncepata, te pronalazi interpretaciju koja zadovoljava koncepte u pitanju. Reduciranje problema inkluzije temelji se na dokazivanju ako je C ⊑D tada C⊓¬D ne može vrijediti. Ovo je intuitivno jasno. Ako je C uključen u D, tada presjek C i neD ne može biti zadovoljen zato što je C cijeli uključen u D. Na sličan način kod reduciranja problema ekvivalencije ne mogu vrijediti dva uvjeta: C ⊓¬D i D ⊓¬C.

Primjer tableau algoritma

Zadano je: `Muz ⊑ Muskarac, Muskarac ⊑ Osoba`

Pitanje: Je li `Muz ⊑ Osoba`

Rasuđivanje: Testiramo postoji li pojedinac koji je `Muz` i nije `Osoba`, odnosno testiramo zadovoljenje koncepta K_0= (Muz⊓¬Osoba):

K_0(a) → (Muz(a)⊓¬Osoba(a))

Muz(a) → Muskarac(a)

Muskarac(a) → Osoba(a)

iz čega ulančavanjem slijedi `Muz(a) → Osoba(a)` što vodi u konflikt s konceptom K_0 da postoji pojedinac koji je `Muz` i nije `Osoba`, pa K_0 nije zadovoljen, a iz toga slijedi da je `Muz ⊑ Osoba` istinito za sve pojedince.

Osim jednostavne $\mathcal{ALC}$ opisne logike koja uvodi atributivne koncepte, u upotrebi su i složenije opisne logike. Na primjer, ako se $\mathcal{ALC}$ proširi s tranzitivnim ulogama, opisna logika postaje *S* logika, a dodavanjem i dodatnih svojstava dobijemo složenije opisne logike na kojima se temelji jezik OWL za opis ontologija koje spominjemo u sljedećem poglavlju. Najvažnija proširenja svojstava prikazuje tablica 4-8 s primjerima:

Tablica 4-8. Osnovna proširenja ALC logike (dodaci na tablicu 4-7)

DL LOGIKA	PROŠIRENJE	PRIMJER
S	tranzitivnost uloga	R* (ivan,petar):imaBrata,(petar,luka):imaBrata → (ivan,luka):imaBrata „Ako Ivan ima brata Petra, a Petar ima brata Luku onda i Ivan ima brata Luku."
$\mathcal{H}$	inkluzija (hierarhija) uloga (podskup)	R ⊑ P imaOca ⊑ imaRoditelja (josip,luka):imaOca→(josip,luka):imaRoditelja „Ako Josip ima oca Luku , onda Josip ima i roditelja Luku, ali ne samo njega, Josip ima i nekog drugog roditelja."
O	nominalni koncepti	Vikend = {Subota,Nedjelja} Roditelji = {Otac,Majka}

[66] Za detalje vidi (Baader et al., 2007).

$\mathcal{I}$	inverzija uloga	R⁻ imaOca ≡ imaDijete⁻ (josip,luka):imaOca ≡ (luka,josip):imaDijete „Ako Josip ima oca Luku, onda Luka ima oca Josipa i obrnuto."
$\mathcal{Q}$	kvalificirana brojevna ograničenja (=, =<, >=)	P ⊑ >= nR.C Bogatas ⊑ >= 2 posjeduje.Kuce „Bogataši su dio skupine vlasnika dvije ili više kuća."
$\mathcal{N}$	brojevna ograničenja (=, =<, >=)	P ⊑ =< nR Doktor ⊑ =< 2 imaRoditelja „Dio onih koji imaju 2 ili manje roditelja su doktori."
$\mathcal{F}$	funkcionalna svojstva	1=< R
$\mathcal{R}$	kompleksna inkluzija (hierarhija) uloga	R ∘ P ⊑ R ili R ∘ P ⊑ P prijateljOd ∘ neprijateljOd ⊑ neprijateljOd „Prijatelj od neprijatelja je neprijatelj."
$\mathcal{D}$	uvođenje tipova podataka	uvode se tipovi podataka tipa int, string, float, boolean i njihovih vrijednosti ∃imaGodina.>=$_{18}$

4.4.6 Ontologije

Ontologije su u informacijskim znanostima primarno razvijene u okviru semantičkog Weba za formalno definiranje kategorija, svojstava i relacija između koncepata, entiteta i podataka unutar određene domene i to prije svega u svrhu razmjene znanja između različitih računalnih entiteta (agenata). Ontologije nisu uvedene kao strukturni oblik pohrane znanja s ciljem formiranja baza znanja o čemu u ovom poglavlju govorimo, ali budući da ontologije ipak predstavljaju strukturni način formalnog prikaza znanja specifične domene, ipak zaslužuju kratki opis. Ovo pogotovo vrijedi zbog toga što se kod projektiranja inteligentnih sustava obično najprije kreće od definiranja ontologije domene u kojoj će inteligentni sustav djelovati, pa su zbog toga ontologije u stvari preduvjet kod definiranja baze znanja.

Prije svega trebamo naglasiti da se ***filozofski pojam ontologije*** bitno razlikuje od ***informatičkog pojma ontologije***. Filozofski pojam ontologije vezan je uz izučavanje pojma ***bitka***. Informacijska znanost samo je posudila pojam i ontologiju definirala kao „***Eksplicitnu specifikaciju konceptualizacije domene***". Jednostavnije kazano, ontologija u informatici definira ***koncepte*** (objekte), kako *opće* tako i *pojedinačne* (individualne), njihove ***klase***, ***svojstva*** i ***atribute*** uključujući i ***ograničenja*** na njih, ***produkcijska pravila***, ***aksiome*** i ***događaje***, a sve to s ciljem definiranja zajedničkog rječnika domene kako bi se to zajedničko značenje moglo razmjenjivati između ljudi, između računalnih entiteta (softverskih agenata), i između ljudi i softverskih agenata. Definiranjem ontologije omogućava se ***ponovno korištenje*** *(*engl. *Reuse)* domenskog znanja kako bi se izbjegla višestruka definiranja i omogućila interoperabilnost. Definiranjem ontologije razdvaja se domensko znanje od operativnog znanja.

Pojam ontologije uveo je 1990. godine ***David Powers*** u nešto drugačijem kontekstu vezanom uz strojno učenje prirodnog jezika, dok je samu definiciju ontologije kao *„specifikaciju konceptualizacije"* uveo 1993. godine ***Tom Gruber*** u članku *„Prema načelima za dizajn ontologija koje se koriste za razmjenu znanja"* (Gruber, 1993.). Od tada do danas razvijeni su ili modificirani brojni računalni jezici prilagođeni definiranju ontologije. Možemo ih grubo podijeliti u tri grupe:

- ***Jezici temeljeni na logici prvog reda FOL*** (engl. *First Order Logic*). Primjer je jezik *KIF* (engl. *Knowledge Interchange Format*) koji je primarno razvijen, kako mu i naziv kaže, za razmjenu znanja između računalnih sustava.
- ***Jezici temeljeni na logici sustava okvira*** (engl. *Frame Logic*). Primjer su već spomenuti jezici *FRL* (engl. *Frame Representation Language*) i *KL-ONE* (engl. *Knowledge Language One*) koji su postojali i prije 1990. godine, ali su sada prilagođeni i za definiranje ontologija.
- ***Mrežni jezici***, sigurno najznačajniji, razvijeni za korištenje u semantičkom Webu. Primjer su *RDF* (engl. *Resource Description Framework*), *XML* (engl. *eXtensible Markup Language*), *DAML* (engl. *DARPA Agent Markup Language*), i na kraju jedan od najznačajniji jezika za prikaz ontologija *OWL* (engl. *Web Ontology Language*).

Jezik *OWL* je semantički Web jezik, standardiziran od W3 konzorcija[67] s ciljem prikazivanja složenog znanja o objektima, grupama objekata i relacija između njih, ali i mogućnošću rasuđivanja, donošenja novog znanja (zaključaka) na temelju definiranog znanja. *OWL* se u tome bitno razlikuje od mrežnih jezika *XML* i *RDF* koji nemaju tu mogućnost. *OWL* je računalni jezik temeljen na logici na način da ga jednostavno može interpretirati drugi računalni program. U prvoj se je verziji pojavio 2004. godine, a u sadašnjoj verziji *OWL 2* prisutan je od 2012. godine. OWL ima tri podjezika: *OWL Light, OWL DL*[68] i *OWL Full* koji se razlikuju po izražajnosti i efikasnosti postupka rasuđivanja i pripadaju različitim tipovima opisne logike. *OWL Light* je SHIF(D), *OWL DL* SHION(D), a *OWL 2 DL* SROIQ(D). U okviru *OWL 2* predložena su i tri podjezika prilagođena specifičnim primjenama: *OWL 2 QL* – za jednostavnu integraciju s bazama podataka, *OWL 2 EL* – jednostavna varijanta jezika i *OWL 2 RL* – posebno projektiran za rad s produkcijskim sustavima.

U ovom kratkom uvodu u *OWL* zaustavit ćemo se samo na tablici 4-8 koja prikazuje povezanost naredbi u *OWL*-u i njihove sintakse u opisnoj logici *(DL)* i predikatnoj logici prvog reda *(FOL)*, te na nekoliko tipičnih primjera zapisa izraza opisne logike u *OWL* jeziku. *OWL* jezik poznaje nekoliko različitih sintaksi, na primjer funkcionalnu, *OWL2 XML, Manchester, RDF/XML i RDF/Turtle* sintaksu. Mi ćemo u primjerima[69] koristiti *RDF/XML* sintaksu i samo na kraju ukratko se osvrnuti i na neke druge sintakse.

Tablica 4-8. Sintaksa naredbi u OWL XML, OWL Manchester sintaksi, te u opisnoj logici (DL) uz primjere

OWL XML izraz	Manchester izraz	DL sintaksa	PRIMJER
Class	Class	Koncepti C	klasa „Musko"
Individual u OWL notaciji owl:Thing	Individual	Pojedinc i a	član klase „Ivan"

[67] https://www.w3.org/OWL/

[68] Kratica DL dolazi od engleskih riječi *„Description Logic"*, u prijevodu „opisna logika" koju smo spomenuli u prethodnom poglavlju.

[69] Ovdje je dana samo kratka reinterpretacija službenog primjera OWL RDF/XML, a za više detalja čitatelje upućujemo na https://www.w3.org/TR/owl-xmlsyntax/apd-example.html

ObjectProperty	ObjectProperty	Uloge R(a,b)	veze između pojedinaca „Ivan ima dijete Marija"
DataPropertyValue DataValue	DataProperty		veze između pojedinaca i vrijednosti „Ivan je visok 1,8m"
KONSTRUKTORI KONCEPATA			
IntersectionOf	C and D	C⊓D	Osoba and Musko „smrtnik i musko"
UnionOf	C or D	C⊔D	Musko or Zensko „musko ili zensko"
ComplementOf	not C	¬C	not Musko „nije musko"
OneOf	{a b}	{a} ⊔ {b}	Ivan i Marija
Restriction(R allValuesFrom(C))	R only C	∀R.C	imaDijete only Profesor „Svi profesori koji imaju dijete."
Restriction(R someValueFrom(C))	R some C	∃R.C	imaDijete some Profesor „Postoji lijecnik koji ima dijete."
Restriction(R maxCardinality(nC))	R max n C	(=<nR.C)	imaDijete max 3 Profesor „Profesor ima najviše 3 djeteta"
Restriction(R minCardinality(nC))	R min n C	(>=nR.C)	imaDijete min 2 Lijecnik „Liječnik ima 2 i više djece"
Restriction(R ExactCardinality(nC))	R exactly n C	(=nR.C)	imaDijete exactly 2 Vozac „Vozac ima 2 djeteta"
hasValue	R value a	∃R(a)	godinaBerbe value 2013
ONTOLOGIJSKI AKSIOMI			
SubClassOf	SubClassOf	C ⊑ D	Osoba⊑Zivotinja⊓Biped
EquivalentClasses	EquivalentClasses	C ≡D	Covjek≡Osoba ⊓ Musko
SubPropertyOf	SubPropertyOf	R ⊑ P	imaSina ⊑ imaDijete
inverseOf	InverseOf	R ≡ ¬P	imaDijete≡not imaRoditelja
EquivalentProperty	EquivalentTo	R ≡ P	imaSina ≡ imaMuskoDijete

FunctionalProperty	FunctionalProperty	R(a,b)≡ R(a,c) b=c	Ako su svojstva rodilaKcer(marija,anu) rodilaKcer(marija,ivu) deklarirana kao funcionalna tada vrijedi: ana = iva
InverseFuncional Property	InverseFuncional Property	R(a,b)≡ R(c,b) a=c	Ako su svojstva rodilaKcer(marija,ana) rodilaKcer(iva,ana) deklarirana kao inverzno funcionalna tada vrijedi: marija = iva
SymetricProperty	SymetricProperty	R(a,b)≡ R(b,a)	Ako je svojstvo imaPrijatelja(ivo,luka) deklarirano kao simetrično tada vrijedi: imaPrijatelja(luka,ivo)
TransitiveProperty	TransitiveProperty	$P^+ \sqsubseteq P$	Ako je svojstvo imaPretka deklarirano kao transitivno tada za P(x,y) i P(x,z) vrijedi P(x,z)

U RDF/XML sintaksi OWL datoteka koja ima ekstenziju .owl započinje i završava `<rdf...>` tagovima između kojih dolazi OWL sadržaj. Pogledajmo primjer definiranja vezene uz opise vina.

```
<owlx:Ontology
     owlx:name="http://laris.fesb.hr/vino"
     xmlns:owlx="http://www.w3.org/2003/05/owl-xml">

<owlx:Class owlx:name="Vino" owlx:complete="false" />
<owlx:Class owlx:name="OpisnikVina" owlx:complete="false" />
<owlx:Class owlx:name="Svijetla" owlx:complete="false" />
<owlx:Class owlx:name="Tamna" owlx:complete="false" />
<owlx:Class owlx:name="CrvenoVino" owlx:complete="false" >
      <owlx:Class owlx:name="#Vino" />
</owlx:Class>
<owlx:Class owlx:name="MirisVina" owlx:complete="false">
      <owlx:Class owlx:name="#OpisnikVina" />
</owlx:Class>
<owlx:Class owlx:name="BojaVina" owlx:complete="false">
      <owlx:Class owlx:name="#OpisnikVina" />
      <owlx:OneOf>
          <owlx:Individual owlx:name="#Bijela" />
          <owlx:Individual owlx:name="#Roza" />
          <owlx:Individual owlx:name="#Crvena" />
       </owlx:OneOf>
</owlx:Class>

<owlx:Individual owlx:name="vinoPlavacMali">
```

```
  <owlx:type owlx:name="CrvenaVina" />
</owlx:Individual>
<owlx:Individual owlx:name="berba2013">
  <owlx:type owlx:name="VinskaGodina" />
  <owlx:DataPropertyValue owlx:property="godinaBerbe">
        <owlx:DataValue
      owlx:datatype="&xsd;positiveInteger">2013</owlx:DataValue>
  </owlx:DataPropertyValue>
</owlx:Individual>

<owlx:ObjectProperty owlx:name=" imaOpisnikVina ">
  <owlx:domain owlx:class="Vino" />
  <owlx:range owlx:class="OpisnikVina" />
</owlx:ObjectProperty>
<owlx:ObjectProperty owlx:name="imaBoju ">
  <owlx:range owlx:class="BojaVina" />
</owlx:ObjectProperty>
<owlx:SubPropertyOf owlx:sub="imaBoju">
  <owlx:ObjectProperty owlx:name="imaOpisnikVina" />
</owlx:SubPropertyOf>

<owlx:Class owlx:name="BijeloVino" owlx:complete="true">
  <owlx:IntersectionOf>
    <owlx:Class owlx:name="#Vino" />
    <owlx:ObjectRestriction owlx:property="#imaBoju">
      <owlx:hasValue owlx:name="#Bijela" />
    </owlx:ObjectRestriction>
  </owlx:IntersectionOf>
</owlx:Class>
</owlx:Ontology>
```

Ontologija je zapis znanja lako razumljiv i čovjeku. U prvoj smo grupi *OWL* naredbi tipa `owlx:Class` definirali nekoliko različitih klasa i pod-klasa. U nastavku smo koristili i inkluzije klasa, npr. `CrvenoVino⊑Vino`. U drugoj smo grupi definirali jednog člana klase (pojedinca) nazvanog `CrvenoVino` koji se zove `PlavacMali`. Naredba za definiranje člana klase `owlx:OneOf` je korištena i kod definiranja klase `BojaVina` na način da su definirana tri moguća člana: `Bijela, Roza i Crvena`. Nakon toga je definirano nekoliko svojstava kod kojih je korištena i definicija domene `owlx:domain` koja definira tko sve može imati svojstvo `owlx:range` koji kaže koje vrijednosti su dopuštene. U slijedećoj je grupi primjer upotrebe presjeka i ograničenja. Klasu `BijeloVino` definiramo kao presjek klase `Vino` i ograničenja na vrijednost svojstva `imaBoju` isključivo na vrijednosti `Svijetla`.

RDF/XML sintaksa slična je *OWL XML* sintaksi, pa su i zapisi slični. *Manchester OWL* sintaksa je sintaksa prije svega prilagođena jednostavnom razumijevanju od strane korisnika i često se koristi u editorima ontologija. Primjer definiranja klase, članova klase i svojstva u ovim sintaksama je:

RDF/XML sintaksa:

```
<owl:Class rdf:ID="Vino" />

<CrnaVina rdf:ID="vinoPlavacMali" />

<owl:ObjectProperty rdf:about="imaOpisnikVina">

            <rdfs:domain rdf:resource="#Vino" />
      <rdfs:range  rdf:resource="#OpisnikVina" />

</owl:ObjectProperty>
```

Manchester sintaksa:

```
Class: Vino

      Annotation: ...
```

```
Individual: PlavacMali
        Annotation: ...
        Type: CrvenaVina ,...
DataProperty: imaOpisnikVina
        Annotation: ...
        Domain: Vino ,...
        Range: OpisnikVina ,...
```

Već smo spomenuli da se kod projektiranja inteligentnih sustava obično kreće od definiranja ontologije domene u kojoj inteligentni sustav treba funkcionirati. Ručna izrada ontologija, pogotovo kod kompleksnijih sustava, nije baš pravo rješenje, već je potrebno koristiti editore ontologija. Jedan od najčešće korištenih editora je ***Protégé***[70], editor otvorenog koda napisan u Javi i razvijen na Stanfordu. Osim desktop varijante Protégé ima i web-verziju[71] koja u potpunosti podržava *OWL 2* i omogućava kolaborativni razvoj ontologija na distribuirani način. Protégé koristi Manchester sintaksu kod definiranja operacija na skupovima i ograničenja svojstava, pa je zbog toga su u Tablici 4.8 dani izrazi i u Manchester sintaksi.

DL logike bazu znanja definiraju skupom aksioma tvrdnji označenih kao ***ABox*** i skupom terminoloških aksioma označenih s ***TBox***.

U OWL-u ***ABox*** čine:

- ***pojedinci (članovi klasa)*** koji predstavljaju konkretne stvari koje mogu pripadati jednoj ili više klasa npr. `vinoPlavacMali, osobaIvan, osobaAna`.

a ***TBox***:

- ***klase*** (skupine pojedinaca koji imaju iste osobine ili ponašanje) poput `Vino, BijelaVina, Osoba, Musko`
- ***hijerarhije klasa*** kojima se definiraju odnosi između klasa poput `subClassOf`
- ***svojstva objekata*** koja definiraju veze između dva pojedinca (člana klase) poput `imaOpisnikVina` i
- ***svojstva podataka*** kojima se pojedincima pridjeljuju konkretne vrijednosti poput `godinaBerbe`.

Kod razvoja ontologija predlaže se sljedeći slijed koji se može ciklički ponavljati:

Analiza zahtjeva i domene → Određivanje svrhe ontologije → Razmatranje mogućnosti ponovne upotrebe → Nabrajanje pojmova → Definiranje klasa → Definiranje svojstava → Definiranje ograničenja → Kreiranje pojedinca

Rasuđivanje u *OWL* jeziku uglavnom se temelji na ***ontologijskom rasuđivanju*** (engl. *Ontology-Based Reasoning*) koje koristi ***zaključivanje temeljeno na klasifikaciji*** (engl. *Classification-Based Inference*). Na primjer, svojstvo `ivan imaZenu marija` uz *obiteljsku ontologiju* povlači i cijeli niz zaključaka koji su uključeni u obiteljsku ontologiju:

`ivan` pripada klasama `Musko, Osoba`

`marija` pripada klasama `Zensko, Osoba`

dok za zaključak tipa

`marija imaMuza ivan`

[70] https://protege.stanford.edu

[71] https://webprotege.stanford.edu

trebalo bi pravilo kojim se povezuju svojstva `imaZenu` i `imaMuza` kao i mehanizam ***rasuđivanje temeljeno na pravilima*** (engl. *Rule-Based Reasoning*) koje nije u potpunosti podržano u *OWL*-u. Postoje određena pravila primjerice `inverseOf`[72] ili `transitiveProperty`, ali za ozbiljnije rasuđivanje temeljeno na pravilima nužno je *OWL* proširiti jezicima pravila. Primjer su tri jezika koja podržava i *w3.org*: ***SWRL*** *(*engl. *Semantic Web Rule Engine)*[73], ***Notation 3 (N3) logic***[74] *i* ***RIF*** *(*engl. *Rule Interchange Format)*[75].

SWRL je podržan u novijim verzijama ***Protégé*** editora kao nadopuna OWL DL jezika. Ima različite sintakse, od formalne XML koja je kombinacija OWL XML prezentacije i RuleML XML jezika do simbolička sintakse lako čitljive čovjeku u kojoj se pravila zapisuju vrlo slično logičkim implikacijskim zapisima. Na ovaj se način pravila unose i u ***Protégé***. Na primjer:

```
Covjek(?p) -> Osoba(?p)

Osoba(?p), imaBracuSestre(?p,?s) -> imaBrata(?p,?s)

Osoba(?x), imaDijete min 1 Osoba(?x) -> Roditelj(?x)
```

Protégé je editor u kojem se uz ontologije mogu unijeti i ovakva pravila, ali za zaključivanje temeljeno na pravilima treba i ***mehanizam zaključivanja korištenjem pravila*** *(*engl. *Reasoner ili Rule Engine)* koji će na temelju ulaznih podataka i pravila donijeti zaključak:

```
ivan imaZenu marija

(?a imaZenu ?b) -> (?b imaMuza ?a)

MEHANIZAM ZAKLJUČIVANJA:

marija imaMuza ivan
```

Postoji puno programa[76] koji imaju ugrađene mehanizme zaključivanja korištenjem pravila. Razlikuju se po tome na koji se jezik pravila naslanjaju, na koji je način realizirano ulančavanje pravila i jesu li komercijalni, slobodni zatvorenog koda ili slobodni otvorenog koda. Primjer programa otvorenog koda je ***Cwm***[77], univerzalni procesor podataka semantičkog Web-a koji ima implementirano unaprijedno ulančavanje i koristi pravila zapisana u jeziku pravila ***Notation 3 (N3) logic***. Tu je i ***Euler/EYE***[78] s ulančavanjem unatrag i pravila zapisana u ***Notation 3 (N3) logic***, a posebno je zanimljiv ***SQWRL*** (engl. *Semantic Query-Enhanced Web Rule Language*)[79] prilagođen korištenju pravila zapisanih u SWRL jeziku pravila koji ima i Java API sučelje i grafičko sučelje koje se može integrirati u Protégé. Dobar prikaz tehnologija semantičkog Web-a je (Khriyenko, 2019).

…………………………………………

Dodatak: Slika[80] ispod prikazuje prof. ***Harolda Cohena*** (1928.-2016.), pionira primjene umjetne inteligencije u umjetnosti i autora prvog računalnog programa ***AARON*** koji je crtao originalne umjetničke slike. Neke od njih prikazana je u desnom dijelu i ispod. AARON se razvijao od monokromatskog *„umjetnika"*, do *„umjetnika"* koji je izvrsno baratao bojama koristeći različite tehnike.

---

[72] U pravilo `inverseOf` mogli bismo uklopiti gornji primjer definirajući `imaMuza inverseOf imaZenu`, te na temelju toga zaključiti `marija imaMuza ivan`, a može se defirati i posebno pravilo kako će se pokazati u nastavku.

[73] https://www.w3.org/Submission/SWRL/

[74] https://www.w3.org/TeamSubmission/n3/

[75] https://www.w3.org/TR/rif-overview/

[76] Popis programa može se pronaći na https://en.wikipedia.org/wiki/Semantic_reasoner

[77] https://www.w3.org/2000/10/swap/doc/cwm.html

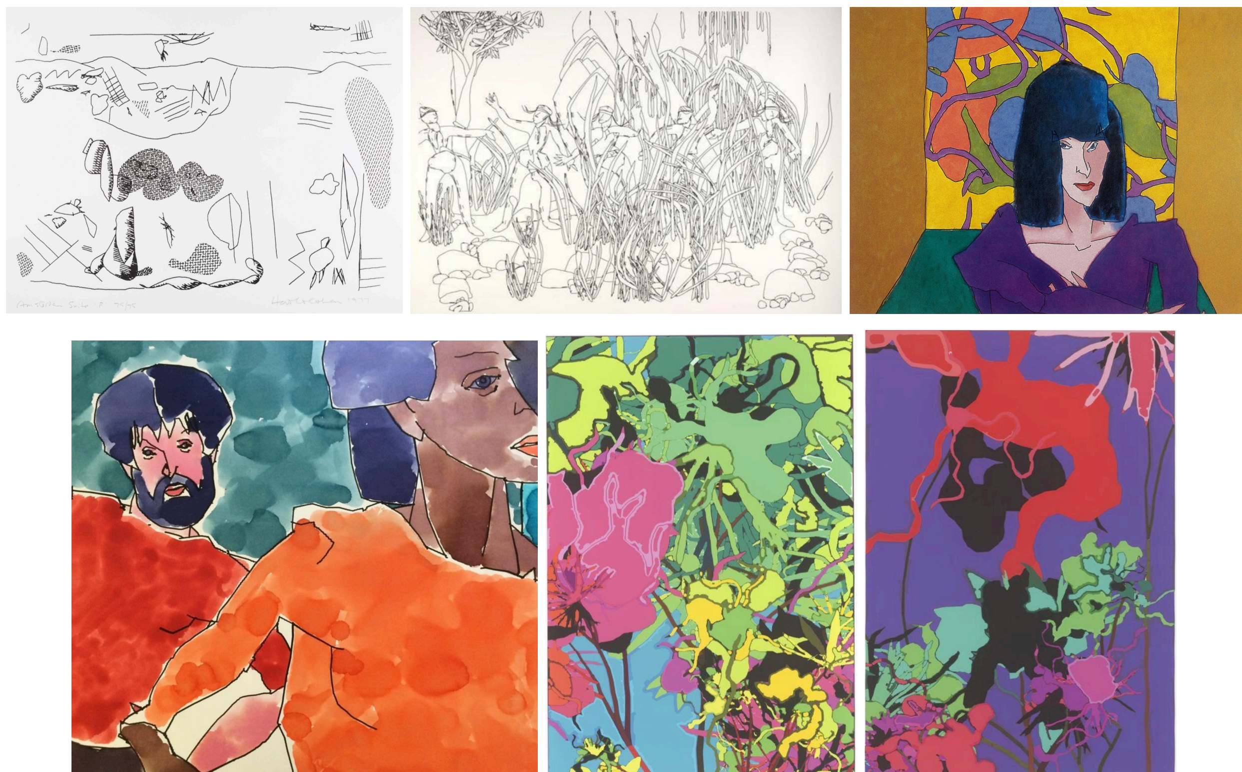
[78] https://github.com/josd/eye

[79] https://github.com/protegeproject/swrlapi/wiki/SQWRL

[80] Slika i Cohenove izjave su preuzeti s http://geneticsandculture.com/genetics_culture/pages_genetics_culture/gc_w05/cohen_h.htm

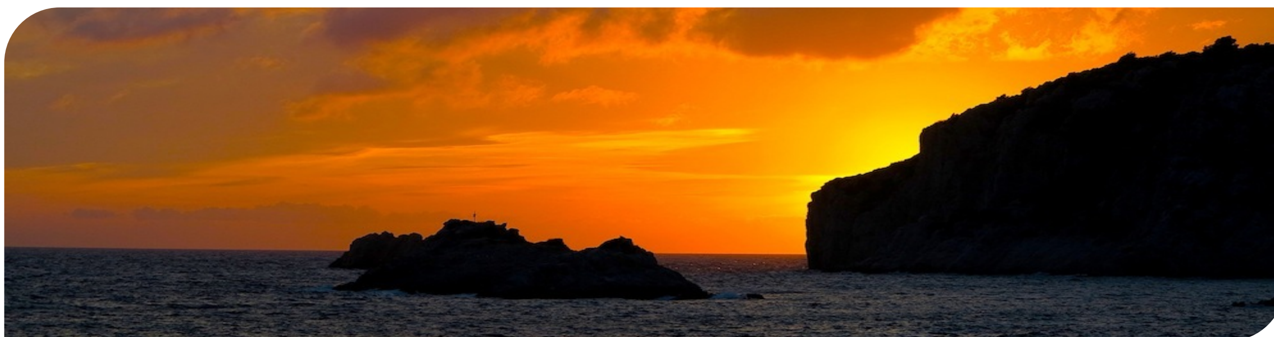


Cohen je u jednom razgovoru kazao: *„Napisao sam ga (misli na AARON-a) kako bih otkrio što bi nezavisna (strojna) inteligencija mogla učiniti, na temelju nekog znanja o svijetu i rudimentarnih fizičkih sposobnosti. I, pritom, da ONA pouči MENE mogućnostima koje nisam zamišljao. Bio bih sretniji da AARON -ov rad u budućnosti što MANJE sliči ljudskim radovima.”*. AARON nije vidio niti jednu svoju sliku. Cohen ga je samo učio principe kompozicije, perspektive, slaganja boja, morfologije objekata koje je crtao i slično, kako je i naglasio: *„Ne govorim mu što da radi. Govorim mu ono što mora znati, a ON sam odlučuje što će učiniti."*. AARON je u najvećem dijelu napisan u LISP-u, a njegove slike su prisutne u brojnim galerijama svijeta uključujući i ***Victoria & Albert Museum*** u Londonu u čijoj su kolekciji i neke od slike ispod (*https://collections.vam.ac.uk/search/?q=Cohen%20Harold*). Više o Haroldu Cohenu i njegovom radu na http://aaronshome.com.



Razvoj Aaonovog crtanja od 1977.g. (gore lijevo), 1987.g. i 1995.g. do (ispod) 1995.g. i 2002.g. (obje)

5 ZAKLJUČIVANJE I RASUĐIVANJE





Rasuđivanje je kompleksni misaoni proces koji se sastoji od više koraka **zaključivanja** kod kojih, na temelju poznatih činjenica i važećih aksioma, dolazimo do novog znanja čiju smo istinitost izveli na temelju istinitosti onoga što smo već znali. Automatsko zaključivanje i rasuđivanje neizostavni je dio umjetne inteligencije, a kako se provodi u različitim logičkim sustavima, opisuje ovo poglavlje.

Zaključivanje (engl. *Inference*) je proces dobivanja novog znanja na temelju postojećeg znanja (poznatih činjenica) i važećih aksioma u jednom koraku. Možda ga najbolje definira filozof ***Imanuel Kant*** *(1724. – 1804.)* koji kaže da je *„zaključivanje proces izvođenja jednog stava iz jednog ili više drugih stavova"*. Zaključivanje je pojedinačni korak kod kompleksnijeg misaonog procesa kojeg nazivamo rasuđivanje.

Rasuđivanje ili ***rezoniranje*** (engl. *Reasoning*) je sustavno zaključivanje na temelju opaženih činjenica i/ili odgovarajućih pretpostavki, najčešće u više koraka zaključivanja. Rasuđivanje je najviši misaoni proces usko povezan s inteligencijom.

Pogledajmo primjer rješavanja problema dvaju vrčeva sa slike 2-1. Zaključivanje se u svakom pojedinom koraku provodi na način da na temelju trenutnog stanja napunjenosti vrčeva ustanovimo koja se pravila mogu primijeniti, te nakon toga primijenimo odgovarajuće pravilo. To je bio primjer primjene osnovnog pravila logičkog zaključivanja ***modus ponens***. Rasuđivanje bi bio cjeloviti proces rješenja problema, traženje puta od početnog stanja do nekog od ciljnih stanja na sustavni način, na primjer primjenom širinskog pretraživanja, kako smo i ilustrirali u opisu rješavanja problema.

Inteligentniji ljudi uspješnije rasuđuju i uče iz novih situacija. Autori navode više različitih načina ljudskog rasuđivanja, kao što su deduktivno rasuđivanje, induktivno rasuđivanje, abduktivno rasuđivanje, rasuđivanje na temelju analogija i intuitivno rasuđivanje, a nama je posebno značajno automatizirano rasuđivanje koje provodi stroj – računalo koristeći formalni program koji oponaša neke od postupaka rasuđivanja.

Deduktivno rasuđivanje (engl. *Deductive Reasoning*) je rasuđivanje o pojedinačnim slučajevima na temelju općih pravila poštujući pravila ispravnog zaključivanja. Na temelju ulaznih premisa koje su istinite donosimo istinit zaključak. Deduktivno zaključivanje uvodi ***Aristotel*** u 4. st. p. n. e., a njegov tipičan primjer ***modus ponens***, koji smo već puno puta spominjali u spomen na Aristotelovog učitelja Sokrata, glasi:

```
Svi su ljudi smrtni. Sokrat je čovjek. ⊢[81] Sokrat je smrtan.
```

[81] Simbol ⊢ naziva se „turnstile" i u logici označava sintaktičku posljedicu. Ako imamo A ⊢ B to znači da se B može dokazati iz A ako koristimo neki formalni logički sustav. S druge strane simbol ⊨ naziva se „double turnstile" (dupli

U deduktivno zaključivanje spadaju i ***modus tollens***:

`Ako je Micika kokoš, onda je ona ptica. Micika nije kokoš. ⊢ Micika nije ptica.`

i ***silogističko zaključivanje***:

`Svi su ljudi smrtni. Svi su Grci ljudi. ⊢ Svi su Grci smrtni.`

Induktivno rasuđivanje (engl. *Inductive Reasoning*) je rasuđivanje o općem pravilu na temelju pojedinačnih slučajeva. Induktivno rasuđivanje nije uvijek valjano, pa zaključak može biti lažan, iako su premise istinite. Induktivno se zaključivanje također u određenoj mjeri pripisuje Aristotelu, ali ga u modernu filozofiju uvodi filozof ***Francis Bacon*** (1561. – 1626.).

Kod deduktivnog zaključivanja na temelju implikacije: „`Ako je A istinito onda su i B i C istiniti.`" i ulazne tvrdnje „`A je istinito.`" zaključili bismo „`B i C je istinito.`", dok bi kod induktivnog zaključivanja postupak bio obrnut: „`B i C je istinito, pa onda i A mora biti istinito.` ". Međutim, ova tvrdnja ne mora uvijek biti istinita pa bi primjerenija tvrdnja bila: „`B i C je istinito, pa onda očekujemo da i A bude istinito.`". Standardni primjer koji se obično spominje je:

`Svi labudovi koje smo vidjeli su bijeli. ⊢ Na temelju toga zaključujemo da su svi labudovi bijeli.`

A ispravan zaključak bi bio: „`Na temelju toga` **`očekujemo`** `da su svi labudovi bijeli.`", zato što možda u budućnosti naiđemo i na nekog labuda koji nije bijele boje. Prvo zaključivanje spada u jednostavnu indukciju, a osim nje postoje i drugi tipovi induktivnog zaključivanja, na primjer ***statistički silogizam***:

`90% gimnazijalaca upisuje fakultet. Mate je završio Gimnaziju.`
`⊢ Mate će se upisati na fakultet.`

koji ne mora nužno biti istinit. Pravilni zaključak bi bio:

`Očekujemo da će se Mate upisati na fakultet.`

Abduktivno rasuđivanje (engl. *Abductive Reasoning*) je ono kod kojeg nismo sigurni jesu li premise istinite, već vjerujemo da su istinite, pa tako i za zaključak samo možemo vjerovati da je istinit. Zaključku se obično dodaju atributi *„najbolji dostupan"* ili *„najvjerojatniji"*. Abduktivno zaključivanje uvodi američki filozof ***Charles Sanders Peirce*** (1839. – 1914.). Možemo slobodno kazati da se u svakodnevnom životu često naslanjamo na abduktivno zaključivanje i rasuđivanje, a ono je bilo i jedan od temeljnih načina zaključivanja poznatog literarnog detektiva ***Sherlocka Holmesa*** kojeg spominjemo i u sljedećem poglavlju. Na primjer: *„Holmes je primijetio na prstu čovjeka svjetliju prugu i zaključio da je čovjek bio oženjen, ali je skinuo vjenčani prsten."* ili *„Holmes je primijetio da otisci vode u smjeru ograde i zaključio da je lopov pobjegao preko ograde, a ne kroz dvorišna vrata."*. I u jednom i u drugom slučaju radi se o abduktivnom zaključivanju, zaključivanju koje se ne temelji na pouzdano provjerenim, istinitim činjenicama, već na pretpostavkama, iako je naravno Sherlock Holmes uvijek u pravu. Vjerojatnosno zaključivanje temeljeno na Bayesovom teoremu je također tipičan primjer abduktivnog zaključivanja, a ovdje možemo uključiti i neizrazito zaključivanje, u biti bilo koje zaključivanje provedeno u situaciji neizvjesnosti ili nesigurnosti. Često se koristi u ekspertnim sustavima, pa je primjerice jedan od prvih ekspertnih sustava medicinske dijagnostike MYCIN koristio kod donošenja zaključaka *„mjeru vjerovanja*" i *„mjeru nevjerovanja*" (engl. *Measure of Belief* i *Measure of Disbelief*), pa možemo smatrati da je i njegovo rasuđivanje bilo abduktivno.

---

turnstile) i označava semantičku posljedicu. Ako imamo A ⊨ B to znači da je B istinit, ako je A istinit. ⊢ dolazi od pojma „istinski uzrokuje" ili „povlači za sobom", a ⊨ od pojma „istinski jednako" ili „implicira". A ⊢ B znači da B formalno slijedi iz A, a A ⊨ B da je B logička posljedica od A, što znači da tablica istinitosti od A koji može biti i složena logička tvrdnja je identična tablici istinitosti od B koji također može biti složena logička tvrdnja. U desna strana formalno slijedi iz lijeve.

Rasuđivanje na temelju analogija (engl. *Analogical Reasoning*) polazi od premisa koje govore o sličnosti članova unutar klase, pa ako jedan član klase ima neko svojstvo, onda ga vjerojatno imaju i drugi članovi klasa. Primjer može biti:

`Sokrat je čovjek i smrtan. Platon je čovjek. ⊢ Platon je smrtan.`

Ovo je varijanta induktivnog rasuđivanja s pojedinca na pojedinca, pa je zbog toga i slabije. Kod induktivnog se zaključivanja kod rasuđivanja koristi veliki broj primjera, a kod rasuđivanja na temelju analogija samo jedan, pa često vodi krivom zaključku. Na primjer:

`Sokrat je smrtan i muško. Kleopatra je smrtna. ⊢ Kleopatra je muško.`

Intuitivno rasuđivanje (engl. *Intuitive Reasoning*) svojstveno je ljudima i ne može se prikazati algoritmom. Temelji se na intuiciji, sudovima koje um sam stvara na podsvjesnoj bazi. Detaljni mehanizmi nam nisu poznati, samo znamo da se zaključci donose na temelju *„osjećaja"*, a ne na temelju racionalnog razmišljanja. Kako nema znanstvenog utemeljenja, intuicija se najčešće shvaća olako, iako je zabilježeno da je odigrala značajnu ulogu u povijesti znanstvenih otkrića. Međutim, kako se ne može automatizirati, nije značajna za umjetnu inteligenciju.

Automatizirano rasuđivanje (engl. *Automated Reasoning*) nije posebna vrsta rasuđivanja, već područje umjetne inteligencije i računarskih znanosti koje se bavi razumijevanjem postupaka rasuđivanja i njihovog prebacivanja u algoritamski i računalni oblik, kako bi računalo moglo samo provoditi zaključivanje i rasuđivanje. Pojavljuje se koncem 50-ih godina pojavom sustava za automatizirano dokazivanje teorema (engl. *ATP – Automated Theorem Proving*) ili automatizirane dedukcije (engl. *AD – Automated Deduction*), a danas je sastavni dio ***mehanizma za zaključivanje*** (engl. *Inference Engine*) sustava umjetne inteligencije kod kojeg se na temelju ulaznih podataka i znanja pohranjenog u bazi znanja, slijedom zaključivanja po principima ulančavanja unaprijed ili unatrag traži rješenje problema.

U nastavku ovoga poglavlja bavimo se različitim postupcima logičkog zaključivanja i rasuđivanja temeljenog kako na klasičnoj tako i na vjerojatnosnoj i neizrazitoj logici.

5.1 Logičko zaključivanje i rasuđivanje

Jedna od osobina inteligentnih bića je usklađenost sa zakonima logike i mogućnost logičkog zaključivanja i rasuđivanja. Ponašanje entiteta koje nije u skladu s logikom ne smatramo inteligentnim.

Logikom se bave filozofija i matematika. Filozofija se bavi proučavanjem apsolutnih istinitosti i metodologijama provjere valjanosti, a matematika daje formalni jezik za izražavanje rečenica te postupke za zaključivanje i rasuđivanje. Logičko zaključivanje i rasuđivanje prije svega se temelji na ***teoriji dokaza*** koja definira mehanizme kojima izvodimo zaključke iz različitih poznatih ulaznih premisa (pretpostavki ili propozicija). Umjetna inteligencija koristi matematičku logiku kao alat.

U umjetnoj inteligenciji, računalni entitet koji izvodi zaključivanje temeljeno na logici naziva se ***logički agent***. Logički agent je računalna implementacija znanja i postupaka rasuđivanja koji rješava problem neke domene problema.

Logički agent je ***agent temeljen na znanju***. To znači da je centralni dio njegove implementacije ***baza znanja***. Znanje o domeni problema prikuplja se u bazu znanja. Baza znanja sastoji se od zapisa (rečenica) izraženih ***formalnim jezikom***, koje se koriste u donošenju zaključaka. Rečenice se izražavaju u ***jeziku za predstavljanje znanja***. Znanje agenta temeljenog na znanju je uvijek ograničeno brojem činjenica.

Standardni zadaci koje mora implementirati agent temeljen na znanju zovu se *TELL* i *ASK*.

TELL ili *„kazati"* je zadatak unošenja nove informacije. *TELL* zadatak zadužen je za dodavanje rečenice u bazu znanja na način da je poveže s već postojećim znanjem.

ASK ili *„upit"* je zadatak dobivanja postojeće informacije iz baze znanja. *ASK* zadatak utjelovljuje ispitivanje baze znanja. Oba zadatka zahtijevaju pokretanje postupka za zaključivanje.

Tipično, ako baza znanja ne sadrži dovoljno informacija za matematički ispravan zaključak vezan uz upit, logički agent zahtijeva TELL operaciju, tj. nadopunu baze znanja informacijama koje nedostaju za postizanje odgovora. Logički agent korištenjem logičkog zaključivanja dolazi do matematički ispravnog zaključka.

Logičko rješavanje problema sastoji se od opisa problema formalnim matematičkim simbolima te upotrebe matematike za rješavanje problema. Matematika potrebna za logičko rješavanje problema sadržana je u formalnom logičkom sustavu, a ovdje ćemo prikazati korištenje propozicijske logike i predikatne logike prvog reda (engl. *FOPL – First Order Predicate Logic*).

Primjeri koji će se koristiti bit će temeljeni na kratkoj priči koju je napisao ***Sir Arthur Conan Doyle*** iz 1891. godine s naslovom ***„Slučaj identiteta"*** (engl. *A Case of Identity*)[82].



Slika 5-1. Sherlock Holmes dočekuje gospođicu Mary – ilustracija iz Stand Magazin autorice Sidney Paget gdje je 1891. godine priča prvi put objavljena[83]

Sažetak priče[84]**:** *„Priča se odnosi na slučaj gospođice* ***Mary Sutherland****, žene koja ima značajan prihod od kamata na fond koji joj je osnovao ujak. Međutim, Mary radi kao tipkačica i živi od zarede tipkačice, a kamatu na fond daje majci i očuhu* ***Jamesu Windibanku*** *koji je samo 5 godina stariji od nje. Zaručena je za mirnog stanovnika Londona gospodina* ***Hosmera Angela****, koji je nedavno nestao.*

Zaručnik gospođice Sutherland osebujan je lik, prilično miran i prilično tajnovit u svom životu. Gospođica Sutherland samo zna da radi u uredu u ulici Leadenhall, ali ništa konkretnije od toga. Sva su joj pisma napisana pisaćim strojem, čak i potpis, a on inzistira na tome da mu ona pisma šalje preko lokalne pošte.

[82] https://etc.usf.edu/lit2go/32/the-adventures-of-sherlock-holmes/347/adventure-3-a-case-of-identity/

[83] Slika je s https://en.wikipedia.org/wiki/A_Case_of_Identity#/media/File:Iden-01.jpg uz licencu 'Public Domain'.

[84] Sažetak je adaptiran s https://en.wikipedia.org/wiki/A_Case_of_Identity

Vrhunac tužne veze dolazi kada gospodin Angel na sami dan vjenčanja pred oltarom napušta gospođicu Sutherland.

Gospođica Sutherland odlazi kod ***Sherlocka Holmesa*** *čemu se protivi njen očuh.*

Holmes, primjećujući cijeli niz činjenica vezanih uz gospođicu Sutherland i njen opis zaručnika, a posebno činjenicu da se zaručnik susreće s gospođicom Sutherland samo dok je njezin mladenački očuh James Windibank poslovno izvan zemlje, vrlo brzo donosi zaključak. Tipkano pismo potvrđuje njegovo uvjerenje. Samo je jedna osoba mogla od ovoga imati korist, očuh gospodin James Windibank. Holmes zaključuje da je zaručnik nestao jednostavnim izlaskom s druge strane kabine kočije.

Nakon što je riješio misterij, Holmes odlučuje da ne kaže svom klijentu gospođici Sutherland rješenje s obrazloženjem da mu ona neće vjerovati s obzirom da živi u zabludi vezanoj sa zaručnikom. Holmes joj je ranije savjetovao da zaboravi gospodina Angela, ali gospođica Sutherland to ne prihvaća i zavjetuje se da će ostati vjerna zaručniku sve dok se ponovo ne pojavi, a najmanje deset godina."

Kako je Holmes došao do zaključka? Pogledajmo postupke logičkog rasuđivanja koji su mu pritom pomogli.

5.1.1 Zaključivanje i rasuđivanje u propozicijskoj logici

Propozicijska logika je logika koja se temelji na sudovima (propozicijama). Obradili smo je u poglavlju 4.3.1. Propozicije su ***činjenice*** koje mogu biti istine ili neistine.

Činjenice koje saznajemo na početku priče su:

- *Gospođica Mary Sutherland dolazi kod Sherlocka Holmesa.*
- *Mary nosi naočale.*
- *Mary ima različite čarape.*
- *Mary Sutherland radi kao tipkačica.*
- *Ima mali imetak koji joj je ostavio ujak.*
- *Živi s majkom i očuhom.*
- *Očuh je samo 5 godina stariji od nje.*
- *Kamate imetka daje roditeljima, živi od zarade tipkačice.*
- *Mary ima zaručnika Angela Hosmera.*
- *Mary se sa zaručnikom sastaje samo kada je očuh na putu.*
- *Zaručnik joj šalje pisma tipkana na pisaćem stroju, uključujući i potpis.*
- *Zaručnik je nedavno nestao.*
- *Očuh se protivi Marynom odlasku kod slavnog detektiva.*

Ove činjenice potrebno je prikazati formalnim jezikom propozicijske logike.

Rečenice propozicijske logike mogu biti atomne ili složene. Atomne rečenice su sudovi (propozicije) kojima pridružujemo vrijednost istinitosti. Složene rečenice gradimo korištenjem operatora. Primjeri rečenica propozicijske logike vezani s našim primjerom su prikazani u tablici 5-1.

Tablica 5-1. Rečenice propozicijske logike vezane s primjerom iz priče „Slučaj identiteta"

mary_lose_vidi	*Atomna rečenica. Može imati vrijednost istina ili nije istina. U slučaju da radimo opis prije opisanog scenarija, ova rečenica imat će vrijednost istina.*
hosmer_je_otet	*Atomna rečenica čiju istinitost još ne poznajemo.*
¬ hosmer_je_dosao_na_vjencanje	*Rečenica koja se sastoji od suda i operatora negacije.*
hosmer_je_otet ∨ hosmer_je_pobjegao	*Rečenica sastavljena od dva suda povezana operatorom disjunkcije*

A = mary_nosi_naocale

B = mary_lose_vidi

A → B

C = mary_ima_razlicite_carape

D = mary_je_stigla_u_zurbi

(B ∧ C) → D

E = maryn_ocuh_ne_zeli_ukljuciti_detektiva

Rečenice složene od više sudova, a izražene jezikom propozicijske logike, imaju svoju vrijednost istinitosti: istina (true, 1) ili laž (false, 0). Ako je izraz rečenica, ne znači da je nužno istinit. Istinitost rečenice ovisi o istinitosti atomnih sudova koji grade rečenicu i o operatorima koji se pojavljuju u rečenici.

Rečenice propozicijske logike mogu biti:

- ***Valjane*** – tautologije (engl. *Valid*)

istinite za sve vrijednosti sudova

mary_lose_vidi ∨ ¬ mary_lose_vidi

- ***Zadovoljive*** – konzistentne (engl. *Satisfiable*)

za neke vrijednosti sudova istinite (ali ne za sve)

mary_nosi_naocale → mary_lose_vidi

- **Ne-zadovoljive** – kontradiktorne (engl. *Unsatisfiable*)

za sve vrijednost sudova je neistinite

mary_nosi_naocale ∧ ¬ mary_nosi_naocale

Zaključivanje se u propozicijskoj logici svodi na **donošenje sudova o rečenicama** (jesu li tautologije, zadovoljive ili nezadovoljive), a može se izvoditi na dva osnovna načina:

- ispunjavanjem tablica istine
- zaključivanje pravilima.

Zaključivanje tablicom istine

Operacije propozicijske logike definirane su tablicama istine (tablica 4-1). U njoj su eksplicitno navedene vrijednosti složenih propozicija povezanih logičkim operatorima. Za svaki složeni izraz propozicijske logike možemo napisati tablicu istine. Razmotrimo sljedeću složenu propoziciju:

((mary_lose_vidi → hosmer_se_lazno_predstavlja) ∨

(hosmer_inzistira_na_sastancima_u_mraku → hosmer_se_lazno_predstavlja) →

((mary_lose_vidi ∧ (hosmer_inzistira_na_sastancima_u_mraku) → hosmer_se_lazno_predstavlja)

Značenje ove rečenice je sljedeće:

- Operator implikacije govori nam da iz lijeve strane slijedi desna strana. Dakle, ako je lijeva, tada je desna.
- Lijeva strana implikacije je rečenica koja u sredini ima operator disjunkcije (veznik *ili*), pa lijeva strana implikacije govori da je istinit ili prvi ili drugi dio.
- Prvi dio lijeve strane je rečenica sastavljena od dviju propozicija povezanih operatorom implikacije. Ova dva suda mogu imati bilo koje vrijednosti, a vrijednost implikacije ovisi o njihovim vrijednostima. Dakle, tvrdimo da iz suda da *„Mary loše vidi"* možemo zaključiti da joj se *„Hosmer mogao lažno predstavljati"*. Ova je rečenica istinita samo ako su vrijednosti sudova u okruženju koji se opisuje, (našem scenariju), takve da je operacija implikacije dana u tablici istine za takve vrijednosti istina.
- Drugi dio lijeve strane također je implikacija dvaju sudova i ovisi o vrijednosti ovih sudova u scenariju.
- Desni dio centralne implikacije je opet implikacija, a govori nam da ako je istina lijeva strana, tada slijedi sud da se *„Hosmer lažno predstavlja"*.
- Lijeva strana ove desne implikacije je rečenica od dva suda povezana operatorom konjunkcije (veznik *i*), koja je istinita onda i samo onda ako su oba suda istinita.

Ova složena rečenica je istinita, ako iz rečenice

((mary_lose_vidi → hosmer_se_lazno_predstavlja) ∨
(hosmer_inzistira_na_sastancima_u_mraku → hosmer_se_lazno_predstavlja))

slijedi

((mary_lose_vidi ∧ hosmer_inzistira_na_sastancima_u_mraku) → hosmer_se_lazno_predstavlja).

Označimo li propozicije

P = mary_lose_vidi

U = hosmer_se_lazno_predstavlja

V = hosmer_inzistira_na_sastancima_u_mraku

možemo napisati tablicu istine za sve moguće kombinacije vrijednosti ova tri suda (tablica 5-2). Iz tablice istine vidimo da svaka interpretacija ovakve rečenice rezultira istinitom rečenicom, te je ova rečenica ***valjana*** *(tautologija)*, tj. uvijek istinita, za bilo koje kombinacije istinitosti sudova *P, U* i *V*.

Tablica 5-2. Tablica istinitosti složene tvrdnje povezane s primjerom gdje 1 znači istina, a 0 laž. Crveno je označena vrijednost istine cijele složene propozicije.

P	U	V	((P→U)	∨	(V→U))	→	((P ∧ V)	→	U)
1	1	1	1	**1**	1	**1**	1	**1**	1
1	1	0	1	**1**	1	**1**	0	**1**	1
1	0	1	0	**0**	0	**1**	1	**0**	0
1	0	0	0	**1**	1	**1**	0	**1**	0
0	1	1	1	**1**	1	**1**	0	**1**	1
0	1	0	1	**1**	1	**1**	0	**1**	1
0	0	1	1	**1**	0	**1**	0	**1**	0
0	0	0	1	**1**	1	**1**	0	**1**	0

Pravila zaključivanja (izvođenja)

Iako nam izvođenje tablice istine daje dobar uvid u značenje složenog logičkog izraza, u slučaju još složenijih rečenica s više sudova i operacija ovakve tablice mogu postati poprilično velike i nepraktične za korištenje. Zbog toga se u tim slučajevima koriste pravila zaključivanja koja složene izraze svode na jednostavnije izraze.

Logička posljedica

G je logička posljedica formule *F1, F2, ... , Fn* ako svaka interpretacija koja zadovoljava *F1, F2, ... , Fn* također zadovoljava i *G*. To formalno pišemo na sljedeći način:

$$F1, F2, \ldots, Fn \vDash G \qquad (5\text{-}1)$$

gdje je ⊨ *„double-turnstile”* simbol koji smo već spominjali u Uvodu ovog Poglavlja, a koji označava semantičku posljedicu.

Na primjer, za situaciju iz naše priče oblika:

$$P1 \wedge P2 \wedge P3 \wedge P4 \vDash Q \qquad (5\text{-}2)$$

mary_lose_vidi ∧ hosmer_inzistira_na_sastancima_u_mraku ∧
hosmer_pise_pisma_na_masini ∧
hosmer_i_mary_sastaju_se_samo_kada_je_ocuh_na_putu

⊨ *hosmer_i_ocuh_su_iste_osobe*

Tablica 5-3 prikazuje zaključivanje korištenjem izravne metode kojom dokazujemo valjanost (tautologiju) i metode opovrgavanja kojom dokazujemo nezadovoljivost (kontradiktornost).

Tablica 5-3. Zaključivanje izravnom metodom i metodom opovrgavanja

Izravna metoda	**Metoda opovrgavanja**
(mary_lose_vidi ∧ hosmer_inzistira_na_sastancima_u_mraku ∧ hosmer_pise_pisma_na_masini ∧ hosmer_i_mary_sastaju_se_samo_kada_je_ocuh_na_putu) → hosmer_i_ocuh_su_iste_osobe Dokazati da je izraz valjan (tautologija).	*(mary_lose_vidi ∧ hosmer_inzistira_na_sastancima_u_mraku ∧ hosmer_pise_pisma_na_masini ∧ hosmer_i_mary_sastaju_se_samo_kada_je_ocuh_na_putu ∧ ¬hosmer_i_ocuh_su_iste_osobe)* Dokazati da je izraz proturječje (kontradiktoran).
Pokazati da je $((F1 \wedge F2 \wedge F3, \dots \wedge Fn) \rightarrow G)$ je valjano.	Pokazati da je $(F1 \wedge F2 \wedge F3, \dots \wedge Fn \wedge \neg G)$ je proturječje.

Kod deduktivnog zaključivanja formula *G* je ***deduktivna posljedica*** formula *F1, F2, … , Fn*, ako i samo ako je *G* moguće ***izvesti*** (engl. *Derive*) iz premisa *F1, F2, … , Fn* ***pravilima zaključivanja*** (engl. *Rules of Inference*). Zaključujemo primjenom jednog od pravila zaključivanja, a izvedenu formulu dodajemo skupu ulaznih pretpostavki (premisa). Postupak ponavljamo dok:

- ***ne dobijemo*** da je izvedena formula identična cilju, i time smo dokazali pretpostavku, ili
- ***više ne možemo izvesti nove logičke formule***, što znači da ne možemo dokazati pretpostavku.

Ovakvo se zaključivanje zove ***prirodno zaključivanje*** (engl. *Natural Deduction*). Postoji više pravila zaključivanja koja u odnosu na operatore koji se u njima pojavljuju možemo podijeliti na pravila konjunkcije, pravila disjunkcije, pravila implikacije i pravila negacije. Svako od njih može biti ***pravilo uvođenja*** (engl. *Introduction Rule*) koje se prema engleskoj riječi označava velikim slovom **I**, na primjer (∧**I**) je *pravilo uvođenja konjunkcije*, ili ***pravilo eliminiranja*** (engl. *Elimination Rule*) koje se označava velikim slovom **E**, pa je na primjer (∧**E**) ***pravilo eliminiranja konjunkcije***. Pravilima uvođenja kombiniramo različite propozicije, a pravilima eliminiranja razbijamo složene propozicije u jednostavnije forme. U nastavku ih sve navodimo.

PRAVILA KONJUNKCIJE

(∧I) - *UVOĐENJE KONJUNKCIJE*

$$P, Q \vdash (P \wedge Q) \tag{5-3}$$

Ovo pravilo kaže da u slučaju kada su dvije propozicije *P* i *Q* svaka za sebe (odvojeno) istinite da će i složena propozicija *(P∧Q)* koju jezično izražavamo *(P i Q)* biti istinita. U zapisu pravila korišten je simbol ⊢ (*turnstile*) koji smo već spomenuli u uvodu kao sintaktičku posljedicu, a jezično se u okviru pravila zaključivanja interpretira riječju ***slijedi*** (engl. *Infer*) pa se zbog toga i pravila zaključivanja ponekad nazivaju i ***pravila slijeđenja (inferencije)***. Drugi način zapisivanja pravila zaključivanja je korištenje simboličkog razlomka koji u brojniku ima pretpostavke, a u nazivniku zaključak. Pravilo uvođenja konjunkcije zapisano na ovaj način izgleda:

$$(\wedge \mathbf{I}) \frac{P, Q}{(P \wedge Q)}$$

Mi ćemo u nastavku koristiti oznaku *turnstile*. Primjer uvođenja konjunkcije je:

mary_ima_imetak, mary_daje_kamate_od_imetka_majci_i_ocuhu

⊢ *(mary_ima_imetak ∧ mary_daje_kamate_od_imetka_majci_i_ocuhu)*

(∧E) – *ELIMINACIJA KONJUNKCIJE*

$$(P \wedge Q) \vdash P \quad (5\text{-}4)$$

$$(P \wedge Q) \vdash Q \quad (5\text{-}5)$$

Ovo pravilo kaže suprotno: u slučaju kada je *(P i Q)* istinito, onda su i propozicije *P* i *Q* svaka za sebe (odvojeno) istinite.

(mary_ima_imetak ∧ mary_daje_kamate_od_imetka_majci_i_ocuhu)

⊢ *mary_ima_imetak, mary_daje_kamate_od_imetka_majci_i_ocuhu*

PRAVILA DISJUNKCIJE

(∨I) - *UVOĐENJE DISJUNKCIJE*

$$P \vdash (P \vee Q) \quad (5\text{-}6)$$

$$P \vdash (Q \vee P) \quad (5\text{-}7)$$

Ovo pravilo kaže da u slučaju kada je propozicija *P* istinita, istinite će biti i propozicije *(P ili Q)* i *(Q ili P)*.

mary_ima_imetak ⊢ (mary_ima_imetak ∨ mary_ima_posao)

mary_ima_imetak ⊢ (mary_ima_posao ∨ mary_ima_imetak)

(∨E) – *ELIMINACIJA DISJUNKCIJE*

$$(P \vee Q), (P \rightarrow R), (Q \rightarrow R) \vdash R \quad (5\text{-}8)$$

Ovo nije jedino pravilo eliminacije disjunkcije ali se obično uzima za standardno. Pravilo kaže da ako vrijedi ako je *P* onda je *R* i ako je *Q* onda je *R*, a uz to znamo da je istinito *P* ili *Q*, onda slijedi da je istinita i propozicija *R*. Drugi primjeri eliminacije disjunkcije su:

$$(P \vee Q), \neg P \vdash Q \quad (5\text{-}9)$$

$$(P \vee Q), \neg Q \vdash P \quad (5\text{-}10)$$

i često se naziva disjunktivno ulančavanje ***(disjunktivni silogizam)***.

vjencanje_se_odrzalo ∨ mary_nema_muža , ¬vjencanje_se_odrzalo

⊢ *mary_nema_muza*

PRAVILA IMPLIKACIJE

(→I) - *UVOĐENJE IMPLIKACIJE*

$$\neg P \vdash (P \rightarrow Q) \quad (5\text{-}11)$$

$$Q \vdash (P \rightarrow Q) \quad (5\text{-}12)$$

Implikacija „*Ako je P onda je Q.*” je istinita ako je *P* lažna, ali isto tako i ako je *Q* istinita.

(→E) – *ELIMINACIJA IMPLIKACIJE*

$$P, (P \rightarrow Q) \vdash Q \quad (5\text{-}13)$$

Ovo je vrlo važno pravilo zaključivanja, možda jedno od najčešće korištenih. Ima i svoje posebno ime i naziva se ***modus ponens***. Ilustrirajmo ga primjerom.

Neka je prva propozicija *P* = *mary_nosi_naocale*, a druga propozicija *Q* = *mary_lose_vidi*. Koristeći modus ponens zaključujemo:

mary_nosi_naocale, mary_nosi_naocale → mary_lose_vidi ⊢ mary_lose_vidi

čije bi jezični opis bio: „*Ako Mary nosi naočale, a naočale nosi kada loše vidi, onda znači da Mary loše vidi.*”.

Uz modus ponens važan je i ***modus tollens***. I ovdje se radi o eliminaciji implikacije, ali na drugačiji način:

$$\neg Q, (P \rightarrow Q) \vdash \neg P \qquad (5\text{-}14)$$

Pogledajmo isti primjer za iste propozicije kao kod *modus ponensa*:

```
⌐ vjencanje_se_odrzalo,hosmer_je_iskren →vjencanje_se_odrzalo
    ⊢ ⌐ hosmer_je_iskren
```

čije bi jezični opis bio: *„Ako nije istina da se vjenčanje održalo, a da je Hosmer iskren vjenčanje bi se održalo, onda znači da nije istina da je Hosmer iskren."*

PRAVILA NEGACIJE

(¬I) - *UVOĐENJE NEGACIJE*

$$(P \rightarrow Q), (P \rightarrow \neg Q) \vdash \neg P \qquad (5\text{-}15)$$

Ovo pravilo zaključivanja naziva se ***reductio ad absurdum*** što možemo prevesti *redukcija do apsurda* ili *redukcija do nemogućeg*. Naziva se i *dokaz nemogućega*, a sam Aristotel ju je često koristio. Ideja polazi od toga da želimo oboriti propoziciju *P*. Ako dokažemo da pretpostavka o istinitosti tvrdnje P vodi u kontradikciju (obje implikacije $(P \rightarrow Q)$ i $(P \rightarrow \neg Q)$ su u tom slučaju istinite), tada je zbog zakona kontradikcije propozicija *P* sigurno lažna.

(¬E) – ELIMINACIJA NEGACIJE

Ponekad se kao eliminacija negacije uvodi već spomenuta ekvivalencija ***dvostruke negacije***.

$$\neg\neg P \vdash P \qquad (5\text{-}16)$$

„Nije istina da Mary nema imetak." vodi u zaključak „*Mary ima imetak.*".

Ovo je ujedno bila jedna od ekvivalencija iz poglavlja 4.3.1. U postupcima zaključivanja koriste se i druge ekvivalencije, na primjer na temelju kontrapozicije zaključujemo:

$$(P \rightarrow Q) \vdash (\neg Q \rightarrow \neg P) \qquad (5\text{-}17)$$

a primjer može biti da na temelju implikacije: *„Ako je Hosmer iskren, onda bi se vjenčanje održalo."* zaključujemo: *"Ako se vjenčanje nije održalo, onda Hosmer nije iskren."*

I na kraju spomenimo još jedan put **ulančavanje** ili **silogizam**. Spomenuli smo ga kod eliminacije disjunkcije kao disjunktivno ulančavanje. Postoji i **hipotetsko ulančavanje** koje se na neki način smatra standardno, pa se izostavlja naziv hipotetsko i naziva se samo ulančavanje (silogizam). Odnosi se na ulančavanje implikacija:

$$(P \rightarrow Q), (Q \rightarrow R) \vdash (P \rightarrow R) \qquad (5\text{-}18)$$

U detektivskom primjeru imali smo situaciju:

„Ako Mary nosi naočale, logično je da ona loše vidi, a ako loše vidi, logično je da Mary ne raspoznaje ljude. Zaključak bi bio ako Mary nosi naočale logično je da Mary ne raspoznaje ljude."

```
(mary_nosi_naocale → mary_lose_vidi),
(mary_lose_vidi → mary_ne_raspoznaje_ljude)
        ⊢ (mary_nosi_naocale →  mary_ne_raspoznaje_ljude)
```

Osnovni nedostatak prirodnog zaključivanja je što je implementacija postupka dosta složena. Upravljački program mora imati dosta složenu upravljačku strukturu koja određuje kada koja pravila zaključivanja upotrijebiti, posebno zbog velikog broja pravila zaključivanja. Postupak se može pojednostavniti uvođenjem novog pravila zaključivanja koje zovemo ***rezolucijsko pravilo*** (engl. *Resolution Rule*) kojim se određenim situacijama može smanjiti broj premisa.

Rezolucijsko pravilo

Krenimo najprije s novom malom detektivskom pričom. Imamo sljedeću situaciju:

„Dogodilo se ubojstvo. Sobarica tvrdi da je u vrijeme ubojstva bila u biblioteci i čistila prašinu, dok susjed tvrdi da ju je čuo u vrtu kako se svađa sa žrtvom kroz otvoren prozor. Dakle, iz sobaričinog iskaza slijedi da je sobarica bila ili u biblioteci ili laže zato što je bila u vrtu. Dok iz susjedovog iskaza možemo zapisati da ili susjed laže zato što sobarica nije bila u vrtu ili je bila u vrtu zato što ju je susjed čuo kako se svađa."

Iz ovoga možemo zaključiti da je „*ili sobarica bila u biblioteci ili je susjed čuo svađu*".

(sobarica_u_biblioteci ∨ sobarica_u_vrtu), (⌐ sobarica_u_ vrtu ∨ susjed_cuo_svađu) ⊢ sobarica_u_biblioteci ∨ susjed_cuo_svadju

U simboličkoj formi ovo zapisujemo:

(A ∨ B), (⌐B ∨ C) ⊢ (A ∨ C)

gdje su:

A = sobarica_u_biblioteci

B = sobarica_u_vrtu

C = susjed_cuo_svadju

Rezolucija svođenjem na konjunktivnu normalnu formu (CNF)

Konjunktivna normalna forma (engl. *CNF - Conjunctive Normal Form*) je logički izraz koji se sastoji isključivo od konjunkcija (∧) i disjunkcija (∨). CNF izrazi posebno su pogodni za automatsko zaključivanje i automatsko dokazivanje teorema. Svođenje logičkih izraza na konjunktivnu normalnu formu provodi se primjenom pravila prikazanih u tablici 5-4.

Tablica 5-4. Pravila za svođenje logičkih izraza na konjunktivnu normalnu formu s primjerima

Zamjena implikacije	$A \rightarrow B$	$\neg A \vee B$
	sobarica_u_vrtu → susjed_cuo_svadju	⌐sobarica_u_vrtu ∨ susjed_cuo_svadju
Zamjena ekvivalencije	$A \leftrightarrow B$	$(\neg A \vee B) \wedge (\neg B \vee A)$
	sobarica_u_vrtu ↔ susjed_cuo_svadju	⌐sobarica_u_vrtu ∨ susjed_cuo_svadju ∧ ⌐susjed_cuo_svadju ∨ sobarica_u_vrtu
Premještanje negacija	$\neg(\neg A)$ $\neg(A \wedge B)$ $\neg(A \vee B)$	A $\neg A \vee \neg B$ $\neg A \wedge \neg B$

	⌐(sobarica_u_biblioteci ∨ sobarica_u_vrtu)	⌐ sobarica_u_biblioteci ∧ ⌐ sobarica_u_vrtu
Transformacija u konjunkciju	$A \lor (B \land C)$ $(A \land B) \lor C$	$(A \lor B) \land (A \lor C)$ $(A \lor C) \land (B \lor C)$
	sobarica_u_biblioteci ∨ (sobarica_u_vrtu ∧ susjed_cuo_svadju) (sobarica_u_vrtu ∧ susjed_cuo_svadju) ∨ sobarica_u_biblioteci	(sobarica_u_biblioteci ∨ sobarica_u_vrtu) ∧ (sobarica_u_biblioteci ∨ susjed_cuo_svadju) (sobarica_u_vrtu ∨ sobarica_u_biblioteci) ∧ (susjed_cuo_svadju ∨ sobarica_u_biblioteci)

Vratimo se sada našoj priči o Mary i pokažimo kako se provodi zaključivanje svođenjem na konjunktivnu normalnu formu.

Polazimo od dostupne baze znanja:

1. mary_nosi_naocale
2. mary_ima_razlicite_carape
3. ocuh_je_mlad
4. ⌐ mary_nosi_naocale ∨ ⌐ mary_ima_razlicite_carape ∨ mary_je_stigla_u_zurbi
5. ⌐ mary_je_stigla_u_zurbi ∨ ocuh_ne_zeli_detektiva
6. ⌐ ocuh_ne_zeli_detektiva ∨ ocuh_nesto_skriva
7. ⌐ ocuh_je_mlad ∨ ⌐ ocuh_nesto_skriva ∨ ocuh_je_hosmer

Temeljem rečenica iz baze, primjenom pravila dobijemo:

8. [1 i 4] ⌐ mary_ima_razlicite_carape ∨ mary_je_stigla_u_zurbi
9. [2 i 8] mary_je_stigla_u_zurbi
10. [9 i 5] ocuh_ne_zeli_detektiva
11. [10 i 6] ocuh_nesto_skriva

Kao konačni zaključak možemo izvesti:

12. [11 i 3] ocuh_je_hosmer

Propozicijska logika je deklarativna logika. Dozvoljava djelomične informacije o scenariju. Također dozvoljava kompoziciju elementarnih sudova u složene sudove, a značenja rečenica propozicijske logike nezavisna su o kontekstu i o semantici elementarnih sudova. Međutim, u usporedbi s prirodnim jezicima propozicijska logika ima ograničenu mogućnost izražavanja. Osnovni nedostatak je što ne dozvoljava izražavanje o odnosima među objektima. Rješenje ovog nedostatka je, kako smo već naglasili, u upotrebi predikatne logike.

5.1.2 Zaključivanje i rasuđivanje u predikatnoj logici

U poglavlju 4.3.2 napravljen je uvod u predikatnu logiku koja se temelji na predikatima, funkcijama koje, ovisno o argumentima, vraćaju vrijednost istinitosti. Osnovne značajke predikatne logike su:

- ***atomi*** opisani pomoću predikata, npr. *nosi_naocale(mary)*
- ***odnosi među atomima*** opisani pomoću predikata, npr. *zaruceni(mary,hosmer))*.

Predikat nam vraća vrijednost istinitosti. Na primjer, ako je *zaruceni(mary,hosmer)* istinito, onda su stvarno Mary i Hosmer zaručeni. Predikatne tvrdnje ovise o vrijednosti varijable istinitosti koju smo uveli. Npr. *ima_sumnje(X)* znači *„netko nešto sumnja"*. A tko je to, to tek treba pronaći.

Sva pravila propozicijske logike vrijede i za predikatnu logiku, samo što nam u predikatnoj logici sudovi imaju bolje izražena značenja.

Predikatna logika uvodi još i kvantifikatore ∀ – *„za svaki"* i ∃ – *„postoji"*, kojima se generaliziraju logički izrazi, a u ***predikatnoj logici prvog reda*** kvantifikatori se odnose samo na varijable. Primjer upotrebe kvantifikatora je:

∀X posjecuje(X,scherlock) → ima_sumnju(X)

što možemo interpretirati: *„Svatko tko posjećuje Sherlocka Holmesa ima nekakve sumnje."*

Primjeri rečenica predikatne logike vezani s našom pričom su:

Mary loše vidi. - *lose_vidi(mary)*

Mary je zaručena za Hosmera Angela. - *zarucena(mary,hosmer)*

Mary ima očuha. - *∃X ocuh_od(mary,X)*

Neki muškarac se lažno predstavlja kao Hosmer Angel i ne želi da se Mary uda. -

∀X muskarac(X), predstavlja_se_kao(X,hosmer), ne_zeli_da_se_uda(mary,X)

Roditelji osobe koja nije udana dobivaju kamate od nasljedstva.

∀X, ∀Y Roditelj_od(X,Y), Nije_udana_(X), ima_nasljedstvo(X) → dobija_novce(Y)

Zaključivanje u predikatnoj logici prvog reda

Zaključivanje tablicama istine koje je bilo korisno u propozicijskoj logici ne može se primijeniti u predikatnoj logici. Nadalje, izravna metoda i metoda opovrgavanja nepraktične su zbog velikog broja vrijednosti koje varijable mogu poprimiti, pa se zbog toga u predikatnoj logici prvog reda koriste druga pravila zaključivanja.

Pravilo univerzalne specijalizacije

Kod ***pravila univerzalne specijalizacije*** (engl. *Universal Instantation Rule*) svaki element domene može biti zamijenjen s univerzalno specificiranom varijablom:

∀X posjeduje_odgovarajucu_pisacu_masinu(X) → pise_lazna_pisma(X)

posjeduje_odgovarajucu_pisacu_masinu (ocuh)

i onda radimo zamjenu varijable *X*:

X=ocuh

posjeduje_odgovarajucu_pisacu_masinu(ocuh) → pise_lazna_pisma(ocuh)

Ako je vrijednost ove implikacije istinita, i ako je vrijednost lijevog operanda istinita, istinita mora biti i vrijednost desnog operanda, pa donosimo zaključak:

pise_lazna_pisma(ocuh)

Kako bismo dokazali da je objekt koji piše lažna pisma upravo očuh, oslanjamo se na implikaciju iz baze znanja koja univerzalno vrijedi, radimo zamjenu varijable objektom. Nakon toga moramo pokazati da je lijeva strana implikacija istinita, a to radimo dodatnim pretraživanjem baze znanja ili nadopunom baze znanja odgovorom na pitanje.

Skolemizacija

U procesu zaključivanja predikatnom logikom često se koristi postupak koji se zove ***skolemizacija*** (engl. *Skolemization*). Ime je dobila po norveškom matematičaru ***Thoralfu Skolemu*** (1887. – 1963.) koji ju je uveo s ciljem sustavne eliminacije egzistencijalnih kvantifikatora. Varijabla vezana s egzistencijalnim kvantifikatorom zamjenjuje se takozvanim ***Skolem konstantama*** ili ***Skolem funkcijama***. Pogledajmo primjer:

∃X posjeduje_odgovarajucu_pisacu_masinu (X) –skolemizacija–> posjeduje_odgovarajucu_pisacu_masinu (ocuh)

∃X ∃Y nasljednik(X,Y) –skolemizacija–> nasljednik (rodjak_od(Y),Y)
–skolemizacija–> nasljednik(rodjak_od(lord),lord)

U prvom izrazu *X* je zamijenjen Skolem konstantom *ocuh*, a u drugom izrazu *X* je najprije zamijenjen Skolem funkcijom *rođak_od(Y)* pa je nakon toga *Y* zamijenjen Skolem konstantom *lord*.

Zašto ovo možemo napraviti? Kako opravdavamo zamjenu varijable konstantom? Lijeva i desna strana izraza sigurno generalno nisu ekvivalencije, ali ako su lijeva i desna strana podjednako zadovoljene u odnosu na istinitost, onda to možemo napraviti, što znači na primjer za prvi izraz:

Ako *∃X posjeduje_odgovarajucu_pisacu_masinu(X) ≡ T (istina)*

onda *posjeduje_odgovarajucu_pisacu_masinu (ocuh) ≡ T (istina)*

Skolemizacija je važan dio postupka zaključivanja predikatne logike, a za više detalja čitatelja upućujemo na literaturu.

Zaključivanje unaprijed i unatrag

Zaključivanje u predikatnoj logici prvog reda zapravo je realizacija *ASK* zadatka logičkog agenta. Baza znanja sastoji se od rečenica izraženih u jeziku predikatne logike prvog reda koje su istinite. Kada imamo izgrađenu bazu znanja, možemo postaviti pitanje i kroz zaključivanje provjeriti je li taj upit istinit.

Dvije su osnovne metode pretraživanja baze znanja: ulančavanje unaprijed i ulančavanje unatrag. ***Ulančavanjem unaprijed*** (engl. *Forward Chaining*) kreće se od poznatih činjenica, primjenjuju se rečenice koje služe kao grupna pravila te se izvode nove činjenice i provjerava se je li se među novim izvedenim činjenicama nalazi i činjenica koja odgovara upitu. Npr. baza znanja sastoji se od rečenica:

nosi_naocale(mary)

∀X, nosi_naocale(X) → lose_vidi(X)

Postavljamo upit:

lose_vidi(mary)

Ulančavanje unaprijed krenut će od poznatih činjenica *nosi_naocale(mary)* te ih primijeniti na pravila. Zamjenom varijabli s konstantom:

nosi_naocale(mary) → lose_vidi(mary)

Ako je implikacija istina, tada istinitost lijeve strane implicira istinitost desne strane, pa zaključujemo da je *lose_vidi(mary)* istinita činjenica. Zaključivanje unaprijed koriste jezici *CLIPS* i *JESS*.

Ulančavanje unatrag (engl. *Backward Chaining*) kreće od upita i pretpostavke da je upit točan te pretražujući bazu znanja traži rečenice koje podržavaju istinitost takve rečenice. Za prethodni primjer postavljamo upit:

lose_vidi(mary)

Ulančavanje unatrag kreće od pretpostavke da je *lose_vidi(mary)* istinito pa traži dokaze za to. Pravilo *∀X, nosi_naocale(X) → lose_vidi(X)* koristi varijablu, i možemo ga upotrijebiti ako varijablu zamijenimo odgovarajućom konstantom kako bi desna strana odgovarala upitu. Rezultat je:

`nosi_naocale(mary) → lose_vidi(mary)`

Ovime je izraženo da se dokaz da je `lose_vidi(mary)` sastoji od dokazivanja da `nosi_naocale(mary)`, te smo sada u potrazi za istinitošću ove rečenice. Kako baza znanja ovu rečenicu sadrži, dokaz je gotov i pronašli smo potvrdu upita. Jezik *Prolog* kod zaključivanja koristi ulančavanje unatrag.

Obje metode korisne su za zaključivanje korištenjem baze znanja koja se sastoji od činjenica i pravila, no obje metode imaju svoje nedostatke. Ulančavanje unaprijed koristi veći broj činjenica nego što nam je potrebno, dok ulančavanje unatrag povratno pretražuje bazu znanja više puta.

5.1.3 Rasuđivanje zdravim razumom

Predikatna logika prvog reda pogodna je za opise stanja realnog svijeta, no ipak nije potpuna. Pojedinačne rečenice predikatne logike mogu biti istinite ili lažne, no predikatna logika ne sadrži mehanizam za promjenu istinitosti rečenica kroz vrijeme. Drugim riječima, nedostaje joj način izražavanja dinamike i vremena kako bi se moglo izraziti opise scenarija događaja u rečenicama još bližima prirodnom jeziku.

Cilj logičkog rasuđivanja je da bude što sličniji ljudskom rasuđivanju ***zdravim razumom*** (engl. *Common Sense*). Rasuđivanje zdravim razumom donosi sudove o fizičkim svojstvima, svrsi, namjerama ili ponašanju ljudi i objekata, kao i mogućih posljedica njihovog djelovanja i interakcija na temelju osjetilnih informacija i znanja. Zdravorazumsko znanje je znanje koje ljudi, često i nesvjesno, skupljaju cijeli život, a dijeli ga većina ljudi i svakodnevno koristi bez da ga je posebno svjesna. Rasuđivanje zdravim razumom temelji se na ***psihologiji zdravog razuma***, ljudskoj sposobnosti da objasni, a donekle i predvidi, ponašanje i mentalna stanja drugih ljudi, te na ljudskoj sposobnosti da može razumjeti i objasniti pojave i događanja u svijetu oko sebe.

Postoje brojni primjeri uspješne primjene zaključivanja zdravim razumom u umjetnoj inteligenciji, kao što je kvalitativno rasuđivanje, rasuđivanje o taksonomijama, temporalno rasuđivanje. Primjer jedne vrste temporalnog rasuđivanja je mehanizam rasuđivanja koji je 1963. godine osmislio ***John McCarthy*** i nazvao ga ***situacijski račun*** (engl. *Situation Calculus*). Situacijski račun, koji se temelji na teoriji zdravog razuma, je s gledišta mogućnosti izražavanja rečenica prirodnog jezika potpuniji od predikatne logike prvog reda jer dozvoljava i dinamiku. U logičko zaključivanje uvedeno je i vrijeme. Činjenice u situacijskom računu su istinite ako je istina da se dogodio događaj koji ih pokreće i nije se dogodio događaj koji ih zaustavlja. Npr.

```
drzi(sherlock,povecalo) IF
   uzeo(sherlock,povecalo) & ¬ostavio(sherlock,povecalo)
```

Za više detalja o situacijskom računu i rasuđivanju zdravim razumom čitatelja upućujemo na dodatnu literaturu kao što je (Kovalski, 1979.) ili (Reiter, 2001.).

5.2 Vjerojatnosno zaključivanje i rasuđivanje

Iako je standardna logika pogodna za zaključivanje i rasuđivanje u umjetnoj inteligenciji, često nije dovoljna za zaključivanje o događajima. Nudi samo dvije vrijednosti istinitosti. To je možda i dovoljno za situacije kada imamo sigurne podatke, ali u stvarnosti rijetko ćemo imati podatke na koje se možemo u potpunosti osloniti.

Mi ljudi zapravo puno češće razmišljamo u duhu vjerojatnosti:

Je li nebo plavo? Vjerojatno je, ali postoji vjerojatnost da nas oči varaju i da je nebo zapravo neke druge boje iako ga svi ljudi doživljavaju kao plavog.

Razlozi za nesigurnost su:

- ***lijenost*** – teško je prikupiti sve podatke i sve uzročno-posljedične veze potrebne za zaključivanje

- ***nedovoljno teorijsko znanje*** – npr. medicina još ne raspolaže dovoljnim teoretskim znanjem za dijagnostiku
- ***nedovoljno praktično znanje*** – ne možemo napraviti sve potrebne testove (npr. u medicini ne možemo napraviti sve moguće pretrage).

Dobra strana vjerojatnosti jest da ona u jednoj veličini, numeričkoj vrijednosti, zbraja nesigurnosti koje proizlaze iz različitih razloga. Racionalni agent uvijek će odabrati akciju koja ima najveću vjerojatnost da ga dovede do cilja.

Vjerojatnosno zaključivanje temelji se na *a priori*, uvjetnim i *a posteriori* vjerojatnostima koje smo već spominjali u poglavlju o vjerojatnosnoj logici. Pogledajmo detaljnije što one znače.

5.2.1 *A priori* vjerojatnosti

A priori ili ***priorne vjerojatnosti*** su vjerojatnosti događaja, vjerojatnosti da se događaj dogodio. Obično se računaju kao omjer broja razmatranih događaja i ukupnog broja mogućih događaja. Na primjer, kod bacanja novčića možemo promatrati događaje:

pismo – novčić je pao na tlo sa stranom na kojoj se nalazi znamenka prema gore

glava – novčić je pao na tlo sa stranom na kojoj se nalazi slika prema gore.

Označimo s *P(glava)* vjerojatnost događaja *glava*. Imamo dva moguća ishoda događaja (*pismo* i *glava*) od kojeg je jedan promatrani događaja (*glava*).

A priori vjerojatnost računamo kao :

P(glava)=(broj željenih događaja)/(ukupan broj događaja) = 1/2 = 0,5

Ako je *P(glava) = 0,5*, možemo postaviti pitanje kolika je *P(pismo)*?

Zbroj *a priori* vjerojatnosti svih događaja u domeni mora biti 1 pa vrijedi:

P(pismo) = 1 - P(glava) = 1 – 0,5 = 0,5

Ako je *P(sunce) = 0,7*, kolika je vjerojatnost da se dogodi neki drugi događaj osim *sunce P(¬sunce)*?

P(¬sunce) = 1 – 0,7 = 0,3

Promotrimo sada pojave koji imaju više od dva ishoda. Skup svih ishoda ili događaja nazivamo domena.

Domena= {sunce, kiša, snijeg, oblačno}

Recimo da je:

P(sunce) = 0,7

P(kiša)=0,2

P(snijeg)=0,02

Postavimo pitanje kolika je a priori vjerojatnost *P(oblačno)*?

P(oblačno) =1–P(sunce) –P(kiša) –P(snijeg) =1 – 0,7– 0,2 – 0,02 = 0,08

5.2.2 Združena vjerojatnost

Kada imamo događaje u nizu, možemo se zapitati koja je vjerojatnost da dva dana zaredom budu sunčana. Ovakve događaje nazivamo ***združenim događajima***. Označavamo ih s $P(S1 \wedge S2)$ ili najčešće još jednostavnije *P(S1,S2)*, a ovaj posljednji oblik zapisa i mi ćemo ovdje koristiti. Vjerojatnost da će se dva događaja dogoditi zajedno nazivamo ***združena vjerojatnost***. Ako su događaji nezavisni, što znači da pojava jednog nije vezana uz pojavu drugog, združenu vjerojatnost računamo kao umnožak *a priori* vjerojatnosti pojedinačnih događaja (više o nezavisnim događajima u poglavlju 5.2.5):

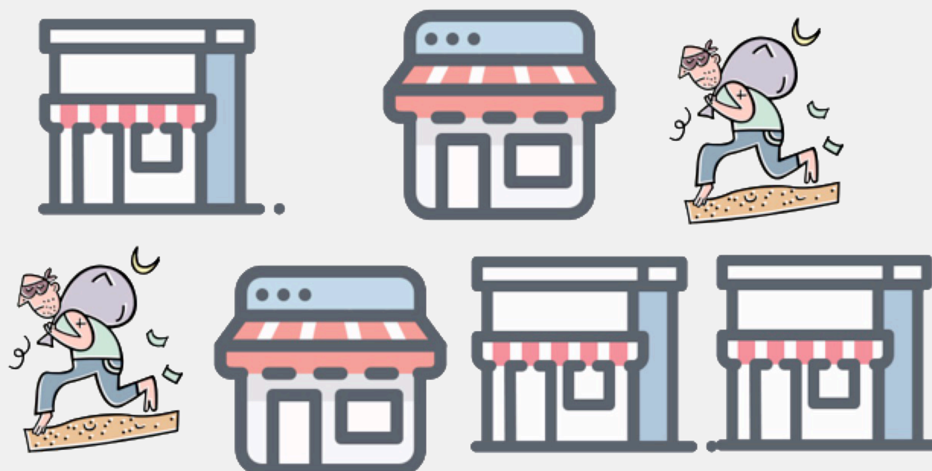
$$P(S1,S2) = P(S1) * P(S2) \quad (5\text{-}19)$$

Na primjer, bacanja novčića su nezavisni događaji, pa možemo izračunati kolika je vjerojatnost da bacanjem dvaju novčića istovremeno na oba bude pismo:

*P(novčić1=pismo i novčić2=pismo) = P(pismo) * P(pismo) = 0,5 * 0,5 = 0,25*

Zadatak 4. – Združene vjerojatnosti

Grad ima 5 trgovina i 2 skupine pljačkaša koji pljačkaju jednom tjedno. Pljačkaši su jednako uspješni i trgovine biraju slučajnim izborom.



Slika 5-2. Dvije skupine pljačkaša i pet trgovina koje se mogu opljačkati

Kolika je vjerojatnost da će trgovina 3 (T3) biti opljačkana barem jednom ovaj tjedan?

P(T3) = 1/5 + 1/5 = 0,4

Kolika je vjerojatnost da će T3 biti opljačkana 2 puta ovaj tjedan?

P(T3,T3) = 1/5 * 1/5 = 0,04

Kolika je vjerojatnost da će skupina 2 opljačkati T2 u petak?

P(P2,T2,D5) = 1/2 * 1/5 * 1/7 = 1/70 = 0,0143

5.2.3 Uvjetna vjerojatnost

Osim združenih vjerojatnosti, možemo promatrati i uvjetne vjerojatnosti koje se označavaju *P(S2 | S1)* i računaju jednadžbom:

$$P(S2 \mid S1) = P(S1, S2) \,/\, P(S1) \qquad (5\text{-}20)$$

gdje je *P(S1, S2*) združena vjerojatnost, a *P(S1) a priori* vjerojatnost događaja *S1*.

Sada se ponovo možemo vratiti združenoj vjerojatnosti i izračunati je preko uvjetne vjerojatnosti ako su događaji *S1* i *S2* zavisni:

$$P(S1, S2) = P(S2 \mid S1) * P(S1) \qquad (5\text{-}21)$$

Ako imamo više od dva događaja *S1, S2, ... , Sn*, primjenjuje se ***pravilo ulančavanja*** (engl. *Chain Rule*):

$$(Sn, \ldots, S1) = \prod_{k=1}^{n} P(Sk \mid \bigcap_{j=1}^{k-1} Sj) \qquad (5\text{-}22)$$

Na primjer za 4 događaja:

$$
\begin{aligned}
P(S4, S3, S2, S1) &= P(S4 \mid S3, S2, S1) * P(S3, S2, S1) = \\
&= P(S4 \mid S3, S2, S1) * P(S3 \mid S2, S1) * P(S2, S1) = \\
&= P(S4 \mid S3, S2, S1) * P(S3 \mid S2, S1) * P(S2 \mid S1) * P(S1)
\end{aligned}
$$

5.2.4 Nezavisnost

Dvije varijable odnosno dva događaja su nezavisna ako pojava jednog ni na koji način ne utječe na pojavu drugog. Ako su *S1* i *S2* nezavisni, tada je njihova združena vjerojatnost (vjerojatnost istovremene pojave obaju događaja) jednaka umnošku njihovih *a priori* vjerojatnosti što smo već napisali u jednadžbi *(5-19)*. Također, poznato je da je za nezavisne događaje uvjetna vjerojatnost jednaka *a priori* vjerojatnosti događaja *S2:*

$$P(S2 \mid S1) = P(S1, S2) \;/\; P(S1) = P(S1) * P(S2) \;/\; P(S1) = P(S2) \qquad (5\text{-}23)$$

Na primjer, zanima nas vjerojatnost da nakon prvog bacanja novčića i drugo bacanje bude pismo.

$P(bacanje1{=}pismo \mid bacanje2{=}pismo) = ?$

Kako su bacanja novčića nezavisni događaji pišemo:

$P(S2 \mid S1) = P(S2) = 0,5$

Uvest ćemo još i pojam ***uvjetna nezavisnost*** (engl. *Conditional Independence*). Kod definiranja uvjetne nezavisnosti ne moramo ništa znati o nezavisnosti dvaju događaja, već ih promatramo samo u prisutnosti trećeg, uvjetnog događaja. Kažemo da su *X* i *Y* uvjetno nezavisni u prisutnosti *Z*. To podrazumijeva zadovoljenje sljedećih uvjeta:

$$P(X,Y \mid Z) = P(X \mid Z) * P(Y \mid Z) \qquad (5\text{-}24)$$

$$P(X \mid Y,Z) = P(X \mid Z) \qquad (5\text{-}25)$$

$$P(Y \mid X,Z) = P(Y \mid Z) \qquad (5\text{-}26)$$

Ako su *X* i *Y* uvjetno nezavisni u prisutnosti *Z*, to znači da pojava *X* ne ovisi o pojavi *Y* i obrnuto (*Y* ne ovisi o pojavi *X*) ako je prisutan *Z*. Pri tome ne mora biti slučaj da su *X* i *Y* nezavisni i bez prisutnosti *Z*. Ovo mora vrijediti za sve vrijednosti varijable *Z*. Drugim riječima, uvjetne vjerojatnosti dviju varijabli *X* i *Y* u prisutnosti *Z* ponašaju se kao nezavisne vjerojatnosti, no bez pojave Z te dvije varijable ne moraju biti nezavisne.

Uvjetna nezavisnost važan je pojam koji se koristi kod grafičkih vjerojatnosnih modela (engl. *Probabilistic Graphical Models*) koji su temelj za d-separaciju. Najpoznatiji primjer grafičkog vjerojatnosnog modela su ***Bayesove mreže*** koje obrađujemo u poglavlju 5.2.6 gdje i detaljnije opisujemo d-separaciju.

5.2.5 Vjerojatnosno zaključivanje

Najjednostavnija metoda za ***vjerojatnosno zaključivanje*** temelji se na kreiranju potpune tablice distribucije združenih vjerojatnosti. Pogledajmo primjer pljačkaša i trgovina. Već smo izračunali da je vjerojatnost da će T3 biti opljačkana 2 puta ovaj tjedan biti:

$P(T3,T3) = 1/5 * 1/5 = 0,04$

a vjerojatnost da će skupina 2 opljačkati T2 u petak:

$P(P2,T2,D5) = 1/2 * 1/5 * 1/7 = 1/70 = 0,0143$

zato što su *a priori* vjerojatnosti pljačkanja bilo koje od 5 trgovina $P(T) = 1/5$, pljačkanja od bilo koje od 2 skupine pljačkaša $P(P) = ½$, i pljačkanja za bilo koji dan u tjednu $P(D) = 1/7$. Ovakav problem možemo rješavati na način da kreiramo tablicu združenih vjerojatnosti koju prilagođavamo po potrebi. Krećemo od definiranja domene problema.

Domena problema:

1. *pljačkaši P1 i P2*

2. *trgovine T1, T2, T3, T4, T5 i*
3. *dani D1, D2 ,D3, D4, D5, D6, D7.*

Ako su sve trgovine jednake, oba tima jednako uspješna u pljačkanju i svi dani u tjednu jednako pogodni za pljačku, tablica se kreira na način da stupci predstavljaju jednu varijablu (dani u tjednu), a redci drugu (trgovine).

Tablica 5-5. Tablica u kojoj definiramo domenu problema

	D1 (ponedjeljak)	D2 (utorak)	D3 (srijeda)	D4 (četvrtak)	D5 (petak)	D6 (subota)	D7 (nedjelja)	sum
T1								1/5
T2								1/5
T3								1/5
T4								1/5
T5								1/5
sum	1/7	1/7	1/7	1/7	1/7	1/7	1/7	1

Tablicu punimo s vrijednostima združenih vjerojatnosti, vodeći pritom računa da je ukupna suma svih vrijednosti u tablici 1. Tako na primjer prva vrijednost tablice na presjeku T1 i D1 predstavlja združenu vjerojatnost da se pljačka T1 dogodi u ponedjeljak. Pritom je vjerojatnost da je T1 opljačkana jednaka 1/5, a vjerojatnost da je pljačka u ponedjeljak 1/7. Množenjem tih dviju vrijednosti dobije se vrijednost združene vjerojatnosti:

$$P(T1 \land D1) = P(T1) * P(D1) = 1/5 * 1/7 = 1/35$$

Na isti se način izračunavaju vrijednosti za sva ostala polja tablice.

Tablica 5-6. Tablica s vrijednostima združenih vjerojatnosti

	D1(pon)	D2 (uto)	D3 (sri)	D4 (čet)	D5 (pet)	D6 (sub)	D7 (ne)	sum
T1	1/35	1/35	1/35	1/35	1/35	1/35	1/35	1/5
T2	1/35	1/35	1/35	1/35	1/35	1/35	1/35	1/5
T3	1/35	1/35	1/35	1/35	1/35	1/35	1/35	1/5
T4	1/35	1/35	1/35	1/35	1/35	1/35	1/35	1/5
T5	1/35	1/35	1/35	1/35	1/35	1/35	1/35	1/5
sum	1/7	1/7	1/7	1/7	1/7	1/7	1/7	1

Nakon što uzmemo u obzir i treća varijablu (skupine pljačkaša S1 i S2), tablica će dobiti treću dimenziju. Za potrebe vizualizacije tablice mi ćemo tu dimenziju prikazati tako da polje podijelimo na dva dijela. Vrijednosti združenih vjerojatnosti su:

$$P(T1 \land D1 \land P1) = P(T1) * P(D1) * P(P1) = 1/5 * 1/7 * 1/2 = 1/70$$

Ovako konstruiranu tablicu možemo koristiti za zaključivanje o vrijednostima združenih vjerojatnosti, ali nam u slučaju jednolike distribucije vjerojatnosti ona neće biti od velike koristi. Međutim, ne moraju svi događaji biti jednako vjerojatni. Uvedimo sada sljedeće iskustvo – na primjer, pljačke su duplo češće petkom. To znači da će u stupcu D5 vrijednosti biti duplo veće od ostalih stupaca. U tom slučaju dobit ćemo nejednoliku distribuciju tako da vrijednosti u stupcu D5 povećamo, ali i u svim ostalim stupcima smanjimo vrijednosti vodeći pritom računa da ukupna suma vjerojatnosti mora ostati 1.

Tablica 5-7. Tablica u kojoj definiramo združene vjerojatnosti dodavanjem i treće varijable

	D1(pon)		D2 (uto)		D3 (sri)		D4 (čet)		D5 (pet)		D6 (sub)		D7 (ne)		sum
	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	
T1	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/5
T2	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/5
T3	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/5
T4	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/5
T5	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/70	1/5
sum	1/7		1/7		1/7		1/7		1/7		1/7		1/7		1

Tablica 5-8. Modifikacije tablice združenih vjerojatnosti uvođenjem iskustvenog pravila da su pljačke duplo češće petkom

	D1(pon)		D2 (uto)		D3 (sri)		D4 (čet)		D5 (pet)		D6 (sub)		D7 (ne)		sum
	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	
T1	1/80	1/80	1/80	1/80	1/80	1/80	1/80	1/80	**2/80**	**2/80**	1/80	1/80	1/80	1/80	1/5
T2	1/80	1/80	1/80	1/80	1/80	1/80	1/80	1/80	**2/80**	**2/80**	1/80	1/80	1/80	1/80	1/5
T3	1/80	1/80	1/80	1/80	1/80	1/80	1/80	1/80	**2/80**	**2/80**	1/80	1/80	1/80	1/80	1/5
T4	1/80	1/80	1/80	1/80	1/80	1/80	1/80	1/80	**2/80**	**2/80**	1/80	1/80	1/80	1/80	1/5
T5	1/80	1/80	1/80	1/80	1/80	1/80	1/80	1/80	**2/80**	**2/80**	1/80	1/80	1/80	1/80	1/5
sum	1/7		1/7		1/7		1/7		1/7		1/7		1/7		1

Uvedimo i novu heuristiku – *pljačkaši P1 preferiraju T2, a pljačkaši P2 češće pljačkaju T4.* Pojedina polja dobivaju veću vrijednost, a ostala manju jer ukupna suma mora uvijek imati vrijednost 1.

Tablica 5-9. Modifikacije tablice združenih vjerojatnosti uvođenjem heurističkih informacija o pljačkama

	D1(pon)		D2 (uto)		D3 (sri)		D4 (čet)		D5 (pet)		D6 (sub)		D7 (ne)		sum
	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2	
T1	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	2/96	2/96	**1/96**	**1/96**	**1/96**	**1/96**	1/5
T2	**2/96**	**1/96**	**2/96**	**1/96**	**2/96**	**1/96**	**2/96**	**1/96**	2/96	2/96	**2/96**	**1/96**	**2/96**	**1/96**	1/5
T3	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	2/96	2/96	**1/96**	**1/96**	**1/96**	**1/96**	1/5
T4	**1/96**	**2/96**	**1/96**	**2/96**	**1/96**	**2/96**	**1/96**	**2/96**	2/96	2/96	**1/96**	**2/96**	**1/96**	**2/96**	1/5
T5	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	**1/96**	2/96	2/96	**1/96**	**1/96**	**1/96**	**1/96**	1/5
sum	1/7		1/7		1/7		1/7		1/7		1/7		1/7		1

Zadatak možemo još dodatno obogatiti novim heuristikama:

- Pljačkaši 1 subotom su u blizini T3, te T3 češće od P2 pljačkaju samo subotom.
- T5 nedjeljom radi kraće tako da je period u kojem može biti opljačkana kraći.

U svakom slučaju, tablica združenih vjerojatnosti čuva nam distribuciju vrijednosti združenih vjerojatnosti za sve vrijednosti i po svim varijablama domene.

Ukupan broj podataka koje ovakva tablica čuva ovisi o broju varijabli i njihovom broju vrijednosti. Za našu tablicu bilo nam je potrebno $5 * 7 * 2 = 70$ podataka. To znači da je za potpunu definiciju tablice združenih vjerojatnosti za ovakav sustav s 3 varijable potrebno individualno pohraniti 70 vrijednosti. S povećanjem broja varijabli eksponencijalno se povećava broj ulaza. Kažemo da imamo ***eksponencijalnu eksploziju*** broja vrijednosti povećanjem broja varijabli. Eksponencijalni rast u računarstvu je uvijek nepoželjan jer se vrlo brzo dosegne količina podataka koja premašuje memoriju računala. Dakle, potrebno je smisliti način kako smanjiti broj varijabli.

Ako još ubacimo i varijablu *sunčan_dan*, tada dobijemo još jednu dimenziju i imamo ukupno $5 * 7 * 2 * 2 = 140$ podataka. Međutim, poznavajući dobro domenu koja u našem slučaju predstavlja običaje pljačkaša, možemo tvrditi da je jednaka vjerojatnost pljačke na sunčan dan kao i na ostale dane (tj. vjerojatnost pljačke ne ovisi o tome je li dan sunčan), te tada informaciju o sunčanom danu možemo isključiti iz naše tablice: $(P, D, T, S) \rightarrow (P,D,T)$, (S) zato što su ti događaji nezavisni.

5.2.6 Bayesove mreže

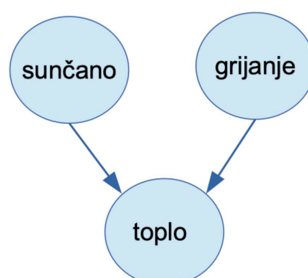
Bayesova mreža je usmjereni graf koji se sastoji od čvorova i lukova, a svaki čvor nosi informaciju o uvjetnim vjerojatnostima. Čvorovi su određeni skupom slučajnih varijabli koje mogu biti diskretne ili kontinuirane. Strelice povezuju dva čvora na način da su strelice usmjerene od čvora roditelj prema čvoru dijete. Svaki čvor nosi informaciju o uvjetnoj vjerojatnosti gdje su uvjeti roditeljski čvorovi $P(X \mid parent(X))$. Graf nema usmjerenih zatvorenih ciklusa, prema tome je ***usmjereni aciklički graf*** (engl. *DAG – Directed Acylic Graph*). Bayesove mreže smanjuju broj varijabli koje je potrebno poznavati da bismo definirali sustav koji opisujemo te su značajno bolje za praktične primjene od tablica združenih vjerojatnosti.

D-separacija

Grafički modeli omogućavaju direktno iščitavanje svojstava uvjetne nezavisnosti iz strukture grafa bez ikakvih analiza, korištenjem okvira koji nazivamo ***d-separacija*** gdje oznaka d dolazi iz engleske riječi *„directed"* što znači usmjerena. Povezanosti u Bayesovim mrežama mogu biti ***konvergirajuće*** (2 uzroka → 1 posljedica), ***divergirajuće*** (1 uzrok → 2 posljedice) i ***serijske*** (uzrok → posljedica1 = uzrok2 → posljedica2).

Konvergirajuća veza

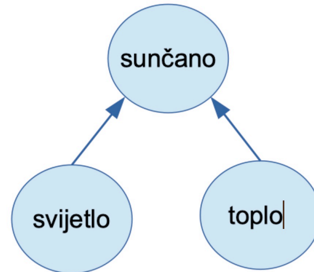
Kod konvergirajuće veze imamo koncept povezanosti gdje više uzroka ima istu posljedicu. Uzroci su nezavisni, kao što je npr. sunčan dan i upaljeno grijanje potpuno nezavisno, ali ako imamo podatak o posljedici, tj. imamo dokaz da je toplo u prostoriji, vjerojatnost da je sunčano i vjerojatnost da je toplo više nisu nezavisne.



Slika 5-3. Konvergirajuća mreža

Divergirajuća veza

Vrijednost varijabli svijetlo i toplina su ovisne. Ako znamo da je svijetlo, veća je vjerojatnost i da je toplo. Ako je mračno, vjerojatnije je da nije toplo. Međutim, ako je dana informacija da je sunčano, vrijednost varijable svijetlo nezavisna je o varijabli toplo i obrnuto. Ovaj primjer predstavlja klasičnu situaciju uvjetne nezavisnosti gdje su varijable *svijetlo* i *toplo* uvjetno nezavisne u prisutnosti dokaza *sunčano*.



Slika 5-4. Divergirajuća mreža

Serijska veza

Serijski vezane varijable su ovisne na primjer za *dobru_fotografiju* treba *svijetlo*, a ono će biti ako je *sunčano*. No ako nam je dana vrijednost varijable *svijetlo*, više nije bitna varijabla *sunčano*, zato što *dobra_fotografija* ne ovisi direktno o njoj.



Slika 5-5. Serijska veza

Primjer Bayesove mreže za profiliranje kriminalaca

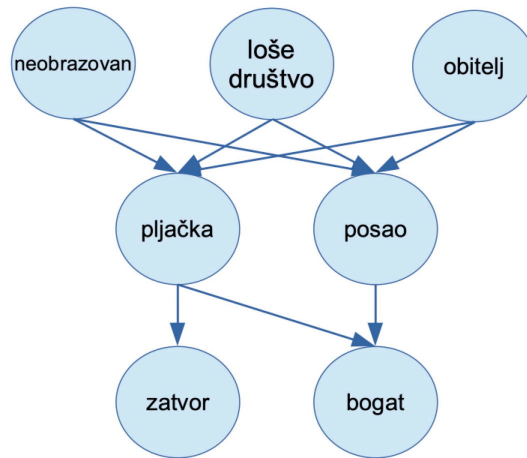
Krenimo primjerom jednostavne Bayesove mreže za profiliranje kriminalaca. Mrežu kreiramo na način da:

- prebrojimo varijable
- povežemo ih vezama
- kreiramo tablice uvjetne vjerojatnosti.

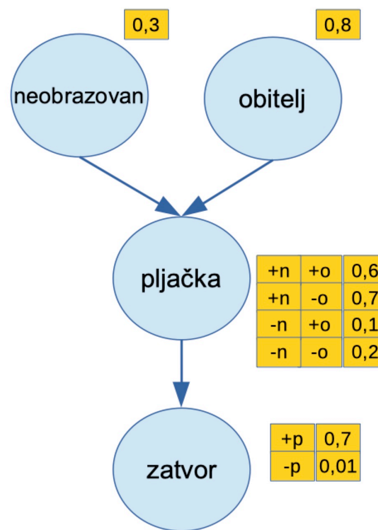
U našem slučaju imamo 7 varijabli, pa bi nam kod kreiranja tablice združenih vjerojatnosti trebalo ukupno 2^7=128 unosa, a za Bayesovu mrežu treba nam samo $1+1+1+2^3 + 2^3+ 2^2+2^2 = 27$ unosa. Pokazat ćemo kako se definira tablica uvjetne vjerojatnosti za dio mreže za profiliranje kriminalaca prikazane na slici 5-7 s tablicom za sve vrijednosti uvjetnih varijabli koje su najčešće samo binarne (1 ili 0). Za opis ovakve mreže potrebno je definirati 2 *a priori* vjerojatnosti i 2 uvjetne vjerojatnosti:

P(neobrazovan), P(obitelj), P(pljačka | neobrazovan, obitelj), P(zatvor | pljačka)

U tablicama + znači da je odgovor potvrdan, a – da je negativan. Na primjer, uvjetna vjerojatnost da se dogodi pljačka ako je pljačkaš neobrazovan (+n) i nema obitelj (-o) je *0,7*, a vjerojatnost da pljačkaš ode u zatvor ako je napravio pljačku (+p) je *0,6*. Na ovoj Bayesovoj mreži ilustrirat ćemo postupak vjerojatnosnog zaključivanja.



Slika 5-6. Bayesova mreža za profiliranje kriminalaca



Slika 5-7. Dio Bayesove mreže za profiliranje kriminalaca sa svim uvjetnim vjerojatnostima

Zaključivanje u Bayesovim mrežama

Zaključivanje ima za cilj za dane ulazne podatke dati odgovor na postavljena pitanja (upite). Na primjer, možemo pitati: *Ako netko ide u zatvor, kolika je vjerojatnost da je neobrazovan?* Rezultat treba biti uvjetna vjerojatnost *P(+n|+z)*. Varijabla *zatvor* je ***varijabla dokaza*** (engl. *Evidence*), a *neobrazovan* je ***varijabla upita*** (engl. *Query*), dok su sve ostale varijable iz modela ***skrivene varijable*** (engl. *Hidden*). Drugi tip pitanja su koje su najvjerojatnije vrijednosti varijabli, ako su im ***dane vrijednosti*** (engl. *Maximum Likelihood*). Na primjer: *Kolika je vjerojatnost da netko ide u zatvor?* Rezultat bi trebao biti *P(+z)*.

Zaključivanje u Bayesovim mrežama radi se raznim postupcima, od kojih ćemo ovdje opisati:

- zaključivanje enumeracijom (nabrajanjem)
- zaključivanje eliminacijom varijabli
- zaključivanje uzorkovanjem.

Zaključivanje enumeracijom (nabrajanjem)

Postavljamo pitanje: *Kolika je vjerojatnost da je netko neobrazovan ako ide u zatvor?*

Prema Bayesovim teoremu:

$$P(+n \mid +z) = P(+n,+z) \ / \ P(+z)$$

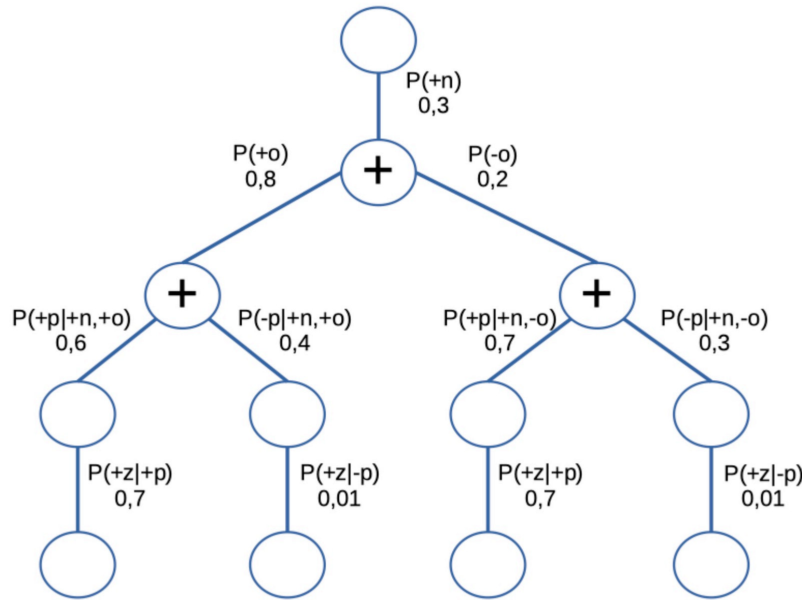
*P(+z)*je ukupna vjerojatnost odlaska u zatvor koja je konstantna, pa je ova uvjetna vjerojatnost proporcionalna združenoj vjerojatnosti:

$$P(+n,+z) = \sum_p \sum_o P(+n,+z,p,o) = \sum_p \sum_o P(+n) \cdot P(o) \cdot P(p \mid +n,o) \cdot P(+z \mid p)$$
$$= P(+n) \cdot \sum_o P(o) \cdot \sum_p P(p \mid +n,o) \cdot P(+z \mid p)$$

Za vrijednosti sa slike 5-7 imamo

P(+n,+z) = P(+n)· (P(+o)· (P(+p | +n, +o)·P(+z | +p) + P(-p | +n, +o)·P(+z | -p)) + P(-o)· (P(+p | +n, -o)· P(+z | +p) + P(+p | +n, –o)·P(+z | –p))) = 0,3 · (0,8 · (0,6 · 0,7 + 0,4 · 0,01) + 0,2 · (0,7 · 0,7 + 0,3 · 0,01)) = 0,3 · (0,3392 + 0,0986) = 0,13134

Postupak enumeracije grafički se prikazuje ***evaluacijskim stablom*** koje za ovaj primjer prikazuje slika 5-8.



Slika 5-8. Evaluacijsko stablo zaključivanja enumeracijom na Bayesovoj mreži sa slike 5-7

Ako je pitanje: *Kolika je vjerojatnost da je netko obrazovan (nije neobrazovan) ako ide u zatvor?*, na sličan način dobijemo:

P(-n,+z) = P(–n) ∑P(o) ∑P(p | – n , o)·P(+z | p) = 0,7 · (0,8 · (0,1 · 0,6 + 0,9 · 0,01) + 0,2 · (0,2 · 0,6 + 0,8 · 0,01)) = 0,7 · (0,0552 + 0,0256) = 0,05656

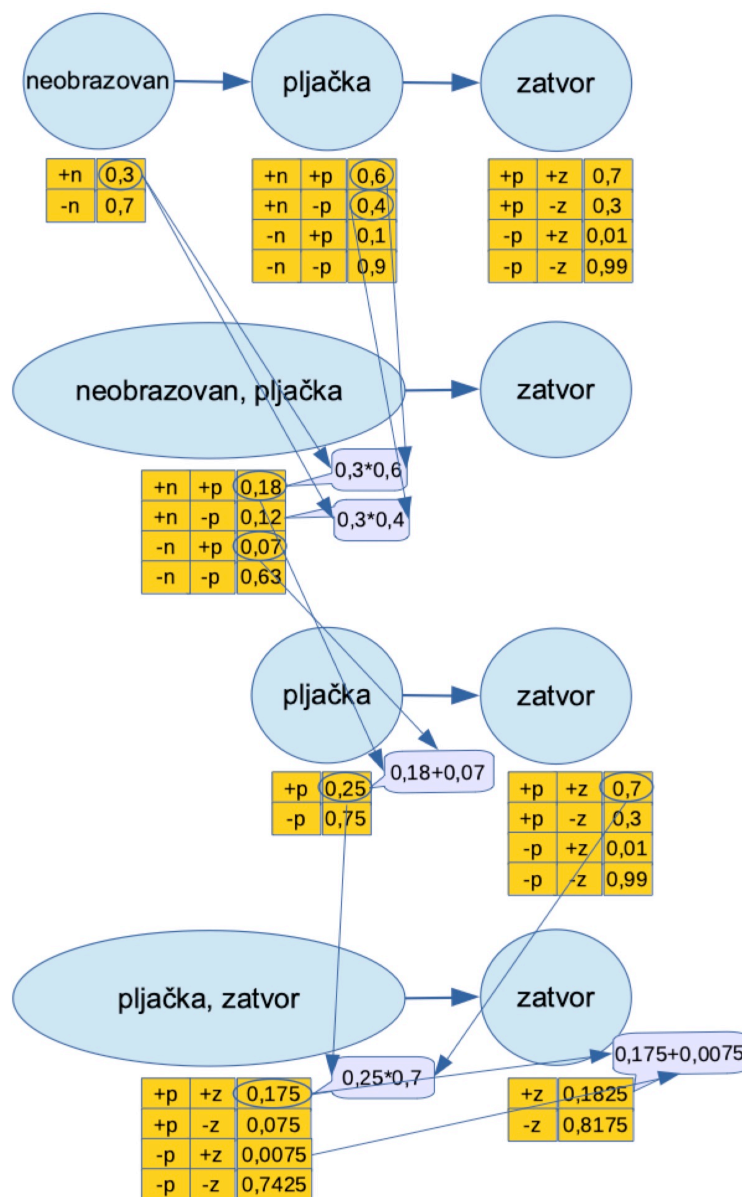
Kako ovo možemo interpretirati? Prema ulaznim podacima neobrazovanih ima 30%, a obrazovanih 70%. Međutim, bez obzira što je obrazovanih duplo više, vjerojatnost da će obrazovani završiti u zatvoru je skoro tri puta manja. Zaključak se temelji na ulaznim podaci o pojedinim vjerojatnostima (slika 5-7), a oni se dobivaju na temelju statističkih analiza stvarnih podataka.

Enumeracija je siguran način zaključivanja u Bayesovim mrežama, ali kod nje broj proračuna može biti velik, pogotovo za složene mreže. Zbog toga su istraživači tražili postupke za ubrzavanje enumeracije. Na primjer, posebnu pažnju treba posvetiti maksimiziranju nezavisnosti kod dizajna Bayesove mreže. S manje veza između čvorova, Bayesova mreža je jednostavnija te proračuni kod zaključivanja imaju manje operacija. Međutim, ukidanjem veza između čvorova možemo propustiti bitne povezanosti među varijablama, Kažemo da prilikom dizajna Bayesove mreže radimo ***balans*** ili ***trgovinu***

(engl. *Trade-off*) između jednostavnosti i preciznosti. Algoritam enumeracije je intuitivan algoritam kojim provodimo zaključivanje u Bayesovim mrežama. On ima svoju matematičku osnovu i sigurno nam daje ispravno rješenje. Međutim, u slučaju složenijih mreža s većim brojem varijabli i gustim vezama, proračuni su neučinkoviti te pribjegavamo drugim rješenjima.

Zaključivanje eliminacijom varijabli

Ponekad prilikom zaključivanja jedan dio mreže koristimo kao skrivene varijable te nam nije potrebno prilikom svakog zaključivanja taj dio mreže ponovo reducirati. Razmotrimo pojednostavljenu Bayesovu mrežu koju prikazuje slika 5-9. Zanima nas kolika je vjerojatnost da osoba završi u zatvoru. Varijable iz mreže eliminiramo tako da združujemo dvije varijable (uzrok i posljedicu) u združenu vjerojatnost dvaju događaja, a zatim računamo vjerojatnost posljedice nezavisno o uzroku. Postupak eliminacije varijabli ilustriran je na slici 5-9.



Slika 5-9. Bayesova mreža kojom ilustriramo algoritam eliminacija varijabli

Zaključivanje uzorkovanjem

Kada analitički postupci ne daju rezultate, možemo pribjeći tehnici uzorkovanja. Prirodni način za određivanje vjerojatnosti pojedinih čvorova jest ***uzorkovanje*** (engl. *Sampling*). Uzmimo za primjer bacanje dvaju novčića. Uzastopnim bacanjem i pamćenjem rezultata (uzorkovanjem) odredit ćemo vjerojatnosti pojedinih ishoda. Preciznost ovakvog zaključka ovisi o broju uzoraka. Što je veći broj uzoraka, to je veća preciznost.

Slično možemo napraviti s Bayesovim mrežama za koje već imamo definirane tablice vjerojatnosti. Uzmimo za primjer Bayesovu mrežu prikazanu na slici 5-7. Postupak započinjemo tako da generiramo uzorak, počevši od varijabli koje nemaju roditelje. Na primjer za *neobrazovan* uzimamo uzorak na način da generiramo slučajnu varijablu, s vjerojatnošću od 0,3 da bude pozitivna. Generiramo varijablu koja je više vjerojatna, a to je da *neobrazovan* nije istinit koji ćemo označiti s *–n*.

Zatim generiramo slučajnu varijablu za *obitelj* i dobijemo pozitivnu vrijednost *+o*. Sljedeća varijabla koju generiramo je *pljačka*. Kako već imamo *–n* i *+o*, iz trećeg retka tablice vidimo da nam varijabla *p* ima vjerojatnost 0,1 da bude pozitivna, odnosno 0,9 da bude negativna. Recimo da dobijemo vrijednost *–p*. Ostaje nam još samo varijabla *zatvor*. Kako imamo ograničenje *–p*, vjerojatnost da dobijemo pozitivnu vrijednost je 0,01, te je veća vjerojatnost da dobijemo vrijednost *–z*. Set generiranih vrijednosti *(–n, +o, –p, –z)* je jedan uzorak. Postupak ponavljamo onoliko puta koliko želimo te iz broja uzoraka možemo izračunati što nam treba.

Ako se upit sastoji od uvjeta, npr. *P(+z | –o)* koji možemo interpretirati *„kolika je vjerojatnost da ide u zatvor ako nema obitelj"*, tada od generiranih uzoraka odbacujemo one koji nisu u skladu s uvjetom, tj. sve koji u sebi sadrže *+o*. Ovo može biti problem, ako se radi o uvjetu koji je manje vjerojatan zato što se velik broj generiranih uzoraka odbacuje, a troše se resursi na njihovo generiranje.

Rješenje je da u svakom uzorku držimo vrijednost uvjeta fiksnim, ali uzorku pridjeljujemo težinu koja je u skladu s ***vjerojatnošću uvjeta*** (engl. *Likelihood Weightning*). Spomenimo još i **Gibbsovo uzorkovanje** kod kojega se svaki novi uzorak razlikuje od prethodnog u samo jednoj varijabli, a varijable dokaza ne diramo.

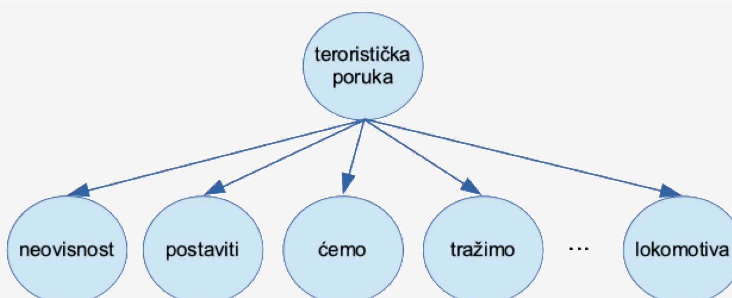
Naivni Bayesov klasifikator

Naivni Bayesov klasifikator (engl. *Naive Bayes Classifier*), ili kako se često zove ***vjerojatnosni klasifikator*** (engl. *Probabilistic Classifier*), je metoda nadziranog strojnog učenja kod kojeg se provodi klasifikacija na temelju Bayesovog teorema uz (naivnu) pretpostavku nezavisnosti između događaja. Strojno učenje detaljno obrađujemo u Poglavlju 6, a ovdje ćemo dati samo kratki uvod u način rada naivnog Bayesovog klasifikatora. Ipak, prije toga ćemo reći par riječi o klasifikatorima i klasifikaciji. Klasifikacija je primjer ***prepoznavanja uzoraka*** (engl. *Pattern Recognition*) i vezana je s opservacijom (opažanjem) podataka o uzorcima i određivanjem kojoj kategoriji pripada neki novoopservirani uzorak. Klasifikacija se kod nadziranog strojnog učenja temelji na skupu podataka za treniranje kod kojeg je za sve uzorke poznato kojoj kategoriji pripadaju. Ako se kod klasifikacije koriste statističke metode, govorimo o statističkom klasifikatoru. Bayesov naivni klasifikator uz korištenje Bayesovog teorema pretpostavlja i statističku nezavisnost između uzoraka. Iz uzoraka se izdvajaju ***značajke*** (engl. *Features*) te se na temelju njih zaključuje kojoj kategoriji uzorak pripada.

Iako se radi o algoritmu strojnog učenja, naivni Bayesov klasifikator utemeljen je na Bayesovim mrežama. Praktičnih primjena ima puno. Na primjer: ***detekcija „smeća"*** (engl. *Spam*) u elektroničkoj pošti, detekcija terorističkih poruka, zaključivanje radi li se o unutrašnjem ili vanjskom prostoru na temelju karakteristika objekata na slici, automatsko prepoznavanje dima šumskog požara na slikama s nadzornih kamera itd.

Zadatak 5. – Naivni Bayesov klasifikator

Način djelovanja naivnog Bayesovog klasifikatora pokazat ćemo na primjeru automatske detekcija terorističkih poruka. Primjer Bayesove mreže za ovaj primjer prikazuje slika 5-10.



Slika 5-10. Struktura Bayesove mreže koju koristi naivni Bayesov klasifikator

Nadzirano strojno učenje podrazumijeva ulaz koji se sastoji od niza podataka kojima je naznačen ispravan izlaz. U slučaju učenja prepoznavanja terorističkih poruka, ulaz je niz poruka od kojih su neke označene kao „*terorističke*", a ostale kao „*ne-terorističke*".

Tablica 5-10. Primjeri terorističkih (T) i ne-terorističkih (NT) poruka

*teroristi**č**ke poruke (T)*	*ne teroristi**č**ke poruke (NT)*
Tražimo neovisnost. Postaviti ćemo bombu.	Pametno postaviti. Tražimo iglu u sijenu. Od igle do lokomotive.

Za odabir značajki koristit ćemo takozvanu „***vreću riječi***" (engl. *Bag of Words*), model koji uzima za značajke pojedine riječi bez obzira na to gdje se u rečenici nalaze i prebrojava broj ponavljanja pojedine riječi.

Tablica 5-11. Rječnik koji pokazuje učestalost pojavljivanja riječi u porukama

tražimo	*neovisnost*	*postaviti*	*ćemo*	*bombu*	*pametno*	*iglu*	*u*	*sijenu*	*od*	*do*	*lokomotive*
2	1	2	1	1	1	2	1	1	1	1	1

Iz ulaznih podataka prvo što možemo izračunati je vjerojatnost pojave terorističke *(T)* i ne-terorističke *(NT)* poruke $P(T)=2/5$, $P(NT)=3/5$. Računamo sada uvjetnu vjerojatnost pojave pojedine riječi, a da se radi o terorističkoj ili ne-terorističkoj poruci:

$$P(bombu\,|\,T) = 1/5$$

$$P(lokomotiva\,|\,T) = 0/5$$

$$P(lokomotiva\,|\,NT) = 1/10$$

Međutim, da bismo realizirali klasifikaciju poruka na terorističke i ne-terorističke, više nas zanima zaključivanje u obrnutom smjeru. Primjerice, možemo izračunati $P(T\,|\,postaviti)$, tj. kolika je vjerojatnost da se radi o terorističkoj poruci ako ona sadrži riječ „*postaviti*". Ova vrijednost predstavlja osnovu klasifikatora, a računa se na sljedeći način:

$$P(T|postaviti) = \frac{P(postaviti|T)*P(T)}{P(postaviti)} = \frac{\frac{1}{5}*\frac{2}{5}}{\frac{2}{16}} = \frac{16}{25}$$

Na isti način možemo izračunati vjerojatnost da je poruka teroristička za sve pojedinačne riječi iz poruke koju klasificiramo. Ukupna vjerojatnost pripadnosti klasi „***terorističkih poruka***" je združena vjerojatnost za sve riječi. Naivni Bayesov klasifikator pretpostavlja da su sve riječi nezavisne jedna o drugoj te se združena vjerojatnost dobije kao umnožak pojedinačnih vjerojatnosti za pojedinačne riječi.

Pogledajmo što se događa kada nas zanima kolika je vjerojatnost da se u terorističkoj poruci pojavi riječ „*bomba*", a kolika da se pojave riječi „*bomba*" i „*lokomotiva*" istovremeno.

$P(T \mid bomba)=P(bomba \mid T) \cdot P(T) / P(bomba) =$

$= P(bomba \mid T) \cdot P(T) / (P(bomba \mid T) \cdot P(T)+ P(bomba \mid NT) \cdot P(NT))=$

$=((1/2) \cdot (2/5))/((1/5) \cdot (2/5)+0 \cdot (3/5)) = 1$

$P(T \mid bomba,lokomotiva)= P(bomba,lokomotiva \mid T) \cdot P(T)/P(bomba,lokomotiva) = P(bomba \mid T) \cdot P(lokomotiva \mid T) \cdot P(T) / (P(bomba \mid T) \cdot P(lokomotiva \mid T)P(T)+ P(bomba \mid NT)P(lokomotiva \mid NT) \cdot P(NT)) = 0$

Vjerojatnost da se u terorističkoj poruci pojavi riječ „*bomba*" je 100%, a da se pojave riječi „*bomba*" i „*lokomotiva*" je 0%. Ipak, znamo da u prirodi rijetko kada imamo događaje kojima je vjerojatnost 100% ili 0%. Sustav ovakvog zaključivanja bio bi nam praktičniji kada bi vrlo vjerojatni događaji imali visoku vjerojatnost, ali vrijednosti manje od 100%, a malo vjerojatni događaji imali ipak neku vjerojatnost veću od 0%. Prisjetimo se da ukupnu vjerojatnost pripadnosti klasi računamo kao umnožak uvjetnih vjerojatnosti pojedinačnih riječi, te znamo da će umnožak s 0 dati konačni rezultat 0, te će samo pojava jedne riječi koja se pojavljuje u zapisima te klase u skupu za treniranje dati ukupnu vjerojatnost 0.

Laplaceovo zaglađivanje (engl. *Laplace Smoothing*) ili aditivno zaglađivanje je postupak koji se koristi za zaglađivanje vjerojatnosti, tj. kako bismo smanjili utjecaj smetnji (tj. ***šuma*** (engl. *Noise*)). Uvedimo najprije pojam ***maksimalne vjerojatnosti*** (engl. *Maximum Likelihood*) koja odgovara maksimalnoj vjerojatnost dobivanja niza podataka.

$P(x) = broj_pojava(x) / broj_uzoraka$

Ovu jednadžbu obično proširujemo Laplaceovim zaglađivanjem:

$P(x) = (broj_pojava(x) + k) / (broj_uzoraka + k \cdot broj_klasa)$

gdje je k ***faktor zaglađivanja*** ($k > 0$, a ako je $k = 0$ nema zaglađivanja). Na primjer, ako imamo jedan uzorak T, dobijemo:

Prema maksimalnoj vjerojatnosti: $P(T) = 1/1 = 1$.

Prema Laplaceovom zaglađivanju za $k=1$: $P(T) = (1+k)/(1+k \cdot 2) = 2/3$.

Na ovaj način dobili smo nove vrijednosti procjene vjerojatnosti događaja temeljem skupa podataka koje su manje podložne utjecaju ekstremnih situacija poput događaja kod kojih nemamo zapisa u skupu podataka ili imamo mali broj uzoraka.

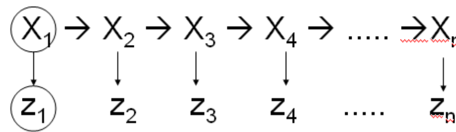
Skriveni Markovljevi modeli

Skriveni Markovljevi[85] ***modeli*** (engl. *HMM – Hidden Markov Models)* koriste se za analizu ili predviđanje promjena stanja kroz vrijeme, pa oni u biti predstavljaju dinamičku Bayesovu mrežu. Zanima nas kako će se stanja razvijati kroz slijed diskretnih trenutaka:

$X1 \rightarrow X2 \rightarrow X3 \rightarrow X4 \rightarrow \dots \rightarrow Xn$

Postoji puno praktičnih primjena skrivenih Markovljevih modela, od robotike, medicine, obrade govora, predviđanja kretanja financijskog tržišta do jednostavnih primjena (npr. u vođenju dizala). Centralni dio skrivenih Markovljevih modela je Bayesova mreža koja se sastoji od niza stanja, a svako stanje ovisi samo o jednom prethodnom stanju i o mjerenjima koji su vidljivi dokazi individualnih stanja. Informacija o stanju su mjerenja z, a stvarna stanja X su skrivena, pa se zato u nazivu nalazi riječ *„skriveni"*. O stvarnim stanjima zaključujemo samo na temelju izmjerenih vrijednosti (slika 5-13).

[85] Ime su dobili po ruskom znanstveniku ***Andreyu Andreyevichu Markovu*** (1856. – 1922.)



Slika 5-13. Kod skrivenih Markovljevih modela o stanjima X_i zaključujemo na temelju mjerenja z_i

Pogledajmo najprije nešto jednostavniji koncept – ***Markovljev lanac*** (engl. *Markov Chain*). Markovljev lanac je niz stanja kod kojeg prelazak u novo stanje ovisi ***samo o prethodnom stanju***. Na primjer: kupci u trgovini plaćaju gotovinom (G) ili karticom (K). Pretpostavimo da vrijedi sljedeće: „*Ako je kupac platio gotovinom, vjerojatnost da će sljedeći platiti gotovinom je 0,6, a karticom 0,4. Ako je kupac platio karticom, sljedeći će kupac s vjerojatnošću 0,8 platiti karticom, dok je vjerojatnost 0,2 da će sljedeći kupac platiti gotovinom.*" Markovljev lanac prikazuje slika 5-14.

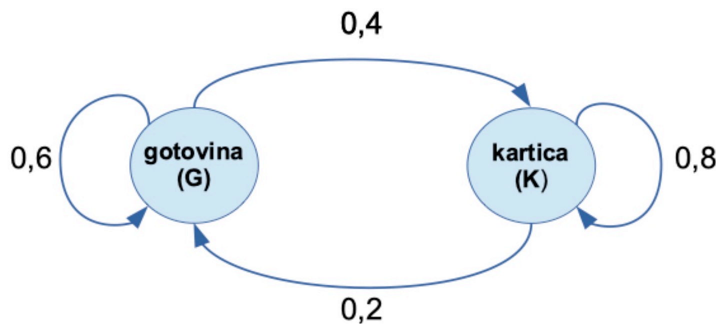
Zanima nas ako je prvi kupac platio gotovinom, kolika je vjerojatnost da će drugi kupac također platiti gotovinom?

$P(G0) = 1$

$P(G1) = ?$

$P(G1) = P(G1 \mid G0) \cdot P(G0) + P(G1 \mid K0) \cdot P(K0) = 0{,}6 \cdot 1 + 0{,}4 \cdot 0 = 0{,}6$

a vjerojatnost da će platiti karticom je $P(K1) = 0{,}4$



Slika 5-14. Markovljev lanac jednostavne situacije plaćanja gotovinom (G) ili karticom (K) u dva plaćanja koja slijede jedan iza drugoga

Kolika je vjerojatnost da će i treći kupac platiti gotovinom?

$P(G2) = ?$

$P(G2) = P(G2 \mid G1) \cdot P(G1) + P(G2 \mid K1) \cdot P(K1) = 0{,}6 \cdot 0{,}6 + 0{,}4 \cdot 0{,}4 = 0{,}52$

Može nas na primjer i zanimati: Postoji li stacionarno stanje, vjerojatnost da će u beskonačnosti kupac platiti gotovinom?

$P(G\infty) = ?$

Za stacionarno stanje vrijedi da je:

$P(Gt) = P(Gt\text{-}1)$

$P(Gt) = P(Gt \mid Gt\text{-}1) \cdot P(Gt\text{-}1) + P(Gt \mid Kt\text{-}1) \cdot P(Kt\text{-}1)$

$X = 0{,}6 \cdot X + 0{,}4 \cdot (1 - X)$

$X = 0{,}5$

Promotrimo sada kako se projektira Markovljev model. U prethodnom primjeru smo pretpostavili da su nam poznate sve vjerojatnosti Markovljevog lanca s slike 5-14. A kako smo do njih došli? Kako popuniti vjerojatnosti u Markovljevom lancu? Jedan od načina je korištenjem maksimalne vjerojatnosti. Za početak nam treba eksperimentalno dobiven vremenski niz. Stali smo pokraj blagajne i bilježili kako kupci plaćaju. Pretpostavimo da smo dobili vremenski niz *GKKKGKG*. Zatim je potrebno popuniti vjerojatnosti prijelaza s onim vrijednostima koje nam pružaju najveću vjerojatnost da za uzorak dobijemo upravo ovakav niz stanja. Iz niza možemo zaključiti sljedeće:

- Ukupno imamo 6 sukcesivnih plaćanja, od toga je kod 2 plaćanja prvo bilo plaćanje gotovinom *(G)*, a kod 4 plaćanje je najprije bilo plaćanje karticom *(K)*. Prvo je plaćanje bilo gotovinom *P(G0)=1*.
- Nakon plaćanja gotovinom, nikada nije u sljedećem plaćanju ponovo plaćeno gotovinom - *P(G|G)=0/2=0*.
- Nakon plaćanja gotovinom uvijek je sljedeće plaćanje bilo karticom - *P(K|G)=2/2=1*.
- Nakon plaćanja karticom u 2 slučaja je sljedeće plaćanje bilo gotovinom, a u dva karticom - *P(G|K)=2/4=0,5* i *P(K|K)=2/4=0,5*.

Kako bi se smanjio utjecaj smetnji, i ovdje možemo koristiti Laplacevo zaglađivanje. Uz $k = 1$ imamo:

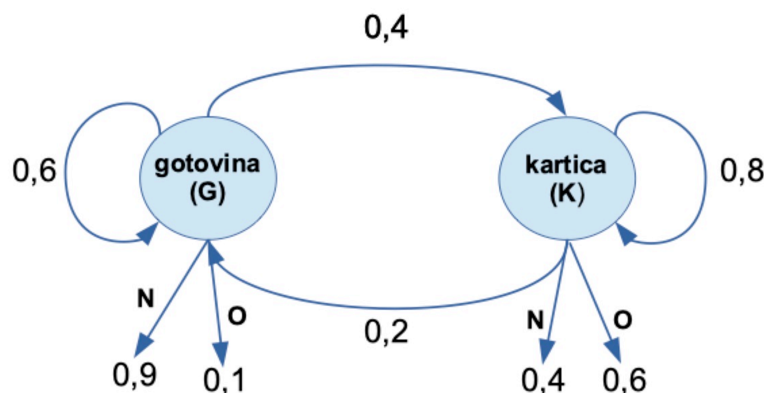
P(G0)=(1+1)/(1+2)=2/3

P(G|G)=(0+1)(2+2)=1/4

P(K|G)=(2+1)/(2+2)=3/4

P(G|K)= P(K|K)=(2+1)/(4+2)=3/6

Vratimo se sada na ***skrivene Markovljeve modele***. Zamislimo da ne možemo izravno vidjeti plaća li kupac gotovinom ili karticom jer nam je kupac okrenut leđima, već vidimo samo trgovca. Međutim znamo da ako kupac plaća gotovinom, trgovac će se nasmijati u 90% slučajeva, dok će se nasmijati u 40% slučajeva kada kupac plaća karticom. Samo stanje nam je skriveno, ali na temelju posrednih podataka (osmijeh trgovca) pokušavamo zaključiti stvarno stanje skrivenih stanja. Model prikazuje slika 5-15.



Slika 5-15. Skriveni Markovljev model situacije plaćanja gotovinom (G) ili karticom (K). N je vjerojatnost da se trgovac osmjehnuo kada je kupac platio gotovinom ili karticom, a O je vjerojatnost da se trgovac nije osmjehnuo, a kupac je platio karticom ili gotovinom.

Ovo je skriveni Markovljev model situacije plaćanja u trgovini pomoću kojega, samo promatrajući osmijehe trgovca, možemo procijeniti stanje gotovine u blagajni. Ako znamo da je *a priori* vjerojatnost plaćanja gotovinom *0,5* i imamo opservaciju da se trgovac nasmijao, možemo izračunati vjerojatnost da je kupac platio gotovinom koristeći Bayesovu formulu.

$P(G|N) = P(N|G) \cdot P(G) / P(N) = (P(N|G) \cdot P(G))/(P(N|G) \cdot P(G) + P(N|K) \cdot P(K))=$

$= (0,9 \cdot 0,5)/(0,9 \cdot 0,5 + 0,4 \cdot 0,5) = 0,7$

$P(G|N)$ je vjerojatnost da je kupac platio gotovinom uz to što se trgovac i nasmijao, $P(N|G)$ je vjerojatnost da će se trgovac nasmijati ako kupac plati gotovinom, a $P(G)$ je *a priori* vjerojatnost da će kupac platiti gotovinom. Zaključak je da je uz osmjeh prodavača povećana vjerojatnost da je kupac platio gotovinom. Skriveni Markovljevi modeli važan su dio algoritama za lokalizaciju robota ili automatske vožnje automobila koji sam vozi.

5.2.7 Klasifikator neosporne konfabulacije

Klasifikator neosporne konfabulacije (engl. *Cogent Confabulation Classifier*) je malo drugačiji od naivnog Bayesovog klasifikatora, a nastao je kao rezultat istraživanja načina kako čovjek zaključuje i rasuđuje. Dio je ***teorije mehanizma razmišljanja*** (engl. *The Mechanism of Thought*) (Hecht-Nielsen, 2005, 2006), a odlikuje ga jednostavna implementacija i brzo računanje. Slično kao i Bayesov teorem neosporna konfabulacija nastoji modelirati induktivno zaključivanje i generalizirati Aristotelovu implikaciju $a,b,c,d \longrightarrow e$, gdje su a, b, c i d četiri različite pretpostavke (premise) na temelju kojih se donosi zaključak e. Ako postoje vjerojatnosti pojave a,b,c,d, uz pretpostavku da će se događaj e dogoditi, vjerojatnost događaja e može se izraziti Bayesovim teoremom kao *a posteriori* vjerojatnost $p(e|abcd)$ definirana jednadžbom:

$$p(e|a,b,c,d) = p(a,b,c,d|e) \cdot p(e)/p(a,b,c,d) =$$
$$= p(a|b,c,d,e) \cdot p(b|c,d,e) \cdot p(c|d,e) \cdot p(d|e)/(p(a|b,c,d) \cdot p(b|c,d) \cdot p(c|d) \cdot p(d)) \qquad (5\text{-}27)$$

U teoriji neosporne konfabulacije pristup je potpuno okrenut, te se traži složena uvjetna vjerojatnost $p(a,b,c,d|e)$ da se uz događaj e pojave i pretpostavke a,b,c,d. U teoriji neosporne konfabulacije $p(a,b,c,d|e)$ *se* naziva ***neospornost*** (engl. *Cogency*). Teorija smatra da se proces donošenja odluka kod ljudi temelji na izboru zaključka koji je najviše podržan istinitim pretpostavkama. Autor teorije Hecht-Nielsen ovu je ideju ilustrirao pričom (Hecht-Nielsen, 2005): „*… ako stvorenje veličine patke sliči na patku, hoda kao patka, pliva poput patke i leti poput patke (premise a,b,c,d), onda ga prihvaćamo kao patku, zato što je patka (e), simbol koji, kad se vidi, najjače podupire vjerojatnost da su te premise istinite i maksimira p(a,b,c,d | e).*"

Bayesov pristup potpuno je drugačiji, zbog toga što on sugerira da se izabere događaj e koji ima maksimalnu *a posteriori* vjerojatnost $p(e|a,b,c,d)$, uz prisutne premise a,b,c,d. Ovo je, prema teoriji neosporne konfabulacije, pogrešan opis ljudskog načina donošenja zaključaka kojeg Hecht-Nielsen ilustrira jednostavnim primjerom danim u nastavku teksta.

Primjer pogrešnog zaključivanja Bayesovim teoremom

Pretpostavimo da za premise a,b,c,d postoje dva moguća zaključka e i f koji imaju *a priori* vjerojatnosti pojavljivanja $p(e) = 0{,}01$ i $p(f) = 0{,}0001$, što znači da je 10 puta veća vjerojatnost da se dogodi e nego f, ili kazano na drugi način, u skupu testnih podataka e se pojavio 10 puta više nego f. Uzmimo kao primjer sliku 4-8 koja prikazuje dim šumskog požara i neka je događaj e = „*pojava kuće na slici*", a događaj f = „*pojava dima na slici*". Prema *a priori* vjerojatnostima na svakih 100 slika na jednoj će se pojaviti kuća, a na svakih 10.000 slika na jednoj će se pojaviti dim. Dim je puno rjeđa pojava nego kuća. Pretpostavimo sada da imamo kombinaciju svojstava a,b,c,d koja se kod pojave e javljaju s uvjetnom vjerojatnosti $p(a,b,c,d\ |e) = 0{,}01$ (na 100 pojava događaja tipa e samo 1 ima kombinaciju a,b,c,d), a kod pojave f s uvjetnom vjerojatnosti $p(a,b,c,d|f) = 0{,}2$ (na sto događaja tipa f njih 20 ima kombinaciju a,b,c,d). Prema Bayesovom teoremu ako na slici primijetimo kombinaciju a,b,c,d vjerojatnost da se radi o zaključku e je čak 5 puta veća od vjerojatnosti da se radi o zaključku f

$$p(e|a,b,c,d)/\,p(f|a,b,c,d) \;=\; \frac{p(a,b,c,d\,|e) \cdot\, p(e)\;/p(a,b,c,d)}{p(a,b,c,d\,|f) \cdot\, p(f)\;/p(a,b,c,d)} = 0{,}01 \cdot 0{,}01/0{,}2 \cdot 0{,}0001 \;=\; 5$$

pa bi po Bayesu, svaki put kada bi se javila kombinacija a,b,c,d, zaključak bio da se radi o pojavi e *(kući)*, što se ne slaže s našim zdravim razumom. Kombinacija a,b,c,d je čak 20 puta češća kod događaja f *(dima)*, ali samo zato što je on rjeđi, Bayesov teorem ne izabire njega, već zaključak e.

Zaključivanje na temelju neosporne konfabulacije izabire onaj zaključak koji ima veću uvjetnu vjerojatnost (neospornost). U ovom su primjeru uvjetne vjerojatnosti $p(a,b,c,d\ |e)$ i $p(a,b,c,d\ |f)$ bile

zadane, pa kako je $p(a,b,c,d \mid f)$ 20 puta veći, za zaključak bi bio odabran *f*, što se i slaže s našim zdravim razumom.

Kod nas su uvjetne vjerojatnosti bile zadane, no u realnoj situaciji one se trebaju odrediti, ali je do njih je vrlo teško doći. Zbog toga se u okviru teorije neosporne konfabulacije stvarna uvjetna vjerojatnost aproksimira jednadžbom:

$$p(a,b,c,d \mid e)^4 \approx K \cdot p(a \mid e) \cdot p(b \mid e) \cdot p(c \mid e) \cdot p(d \mid e) \qquad (5\text{-}28)$$

gdje je *K* konstanta. Jednadžba vrijedi za (naivnu) pretpostavku nezavisnosti između premisa *a, b, c* i *d* i sugerira da će složena uvjetna vjerojatnost (neospornost) biti najveća ako je produkt uvjetnih vjerojatnosti $p(a \mid e) \cdot p(b \mid e) \cdot p(c \mid e) \cdot p(d \mid e)$ najveći. Produkt vjerojatnosti $p(a \mid e) \cdot p(b \mid e) \cdot p(c \mid e) \cdot p(d \mid e)$ se naziva ***konfabulacija*** (engl. *Confabulation*), pa se u biti neospornost $p(a,b,c,d \mid e)$ niti ne računa. Za dva moguća zaključka *e* i *f* računaju se konfabulacije $Con(e) = p(a \mid e) \cdot p(b \mid e) \cdot p(c \mid e) \cdot p(d \mid e)$ i $Con(f)= p(a \mid f) \cdot p(b \mid f) \cdot p(c \mid f) \cdot p(d \mid f)$, te se odabire onaj zaključak čija je konfabulacija veća. Postupak predstavlja primjenu strategije „***pobjednik nosi sve***" (engl. *Winer-takes-all Strategy)* koja je prema teoriji mehanizma razmišljanja upravo način na koji ljudi donose odluke.

Do uvjetnih vjerojatnosti tipa $p(a \mid e)$ ili $p(a \mid f)$ nije teško doći. Na primjeru sa slike 4-8 već smo pokazali kako se npr. računa uvjetna vjerojatnost da neka kombinacija boja na slici pripada dimu, a i u poglavlju 5.2.2 je o uvjetnoj vjerojatnosti dosta kazano. Upravo zbog toga klasifikator neosporne konfabulacije nalazi sve više praktičnih primjena.

5.3 Neizrazito zaključivanje i rasuđivanje

Realnost stvarnog svijeta, problem je koji je stoljećima zaokupljao filozofe i znanstvenike: *Što je realnost? Kako je definirati i opisati?* Činjenica je da je realnost uvijek neprecizna i nesigurna. Ona je najčešće kaotična, dinamička, promjenjiva, nepredvidiva i obično izvan mogućnosti definiranja u obzorima tradicionalnog razmišljanja. Ipak, mi živimo u realnosti i stvaramo u realnosti, pa nas to potiče na razmišljanje kako to da mi ljudi (relativno) uspješno funkcioniramo u takvom okružju.

Kako uspijevamo voziti automobil i prilagoditi se stalno promjenjivoj i nepredvidivoj okolini? Kako uspijevamo svakodnevno djelovati ne poznavajući precizno i detaljno što se oko nas događa? Mi ovisimo o realnosti, realnost je rezultat naših akcija. *I kako onda formalno opisati realnost, kako je uključiti u tehničke i tehnološke sustave koji su u potpunosti proizvod čovjeka, a trebaju djelovati u realnom svijetu? Kako izgraditi sustav vođenja koji će samostalno voditi automobil ili vlak? Kako predvidjeti urod i u slučaju kada nemamo iskusnog poljoprivrednika? Kako procijeniti razvoj vremena i bez iskusnog prognostičara i bez preciznih meteoroloških podataka, automatski, uz pomoć računala na način kako su to stoljećima radili iskusni ribari i pomorci?*

Upravo su ova pitanja 1965. godine navela ***Lotfija Zadeha*** da pokuša objasniti, opisati i definirati nepredvidivost i nejasnost realnog svijeta uvodeći novi hibridni način razmišljanja nazvan ***neizrazita teorija skupova***, a kasnije jednim dijelom i ***neizrazita logika***. Spominjali smo je već u poglavlju 4.3.3 o nestandardnim logikama. Na Zadehovim idejama tijekom proteklih 50-ak godina izrasla je cijela tehnologija i industrija. Ona se danas u potpunosti potvrdila, ne samo kao dobra teorija za opis realnog svijeta, ne samo kao logički sustav koji je (za sada) najbliži logičkom sustavu ljudskog mozga, već i kao izrazito primjenjiva tehnika u brojnim ljudskim djelatnostima. Svjetska industrija bilježi stotine primjena neizrazite logike, od automatskog vođenja rotacione peći za proizvodnju cementa i automatskog vođenja vlaka do predviđanja rizika potresa i burzovnih trendova. Suvremena industrija u turbulentnoj okolini realnog svijeta danas je nezamisliva bez neizrazite logike. U ovom nas dijelu posebno zanima kako se donose zaključci i provodi rasuđivanje u duhu neizrazite logike.

Neizrazito zaključivanje je tehnika računanja bez preciznih matematičkih jednadžbi i formula. Umjesto detaljnog matematičkog definiranja zadataka, neizrazito zaključivanje omogućava opisivanje zadataka koristeći približne i aproksimativne vrijednosti i opise slične onima koje čovjek koristi kada govorom opisuje zadatak drugom čovjeku. Primjerice, precizni opis zadatka prilagodbe brzine automobila bi bio:

„Ako je brzina 105 km/h, počni kočiti 123 m prije raskrižja."

što sigurno nije način kako bi ga čovjek u svakodnevnom, realnom životu opisao. Njegovom načinu izražavanja puno je bliži opis:

*„Ako je brzina **oko 100 km/h**, počni kočiti **stotinjak** metara od raskrižja."*

gdje su ***oko 100 km/h*** i ***stotinjak*** približni, aproksimativni opisi vrijednosti brzine i udaljenosti.

Prva je izjava previše precizna da bi bila primjenjiva u svakodnevnoj ljudskoj komunikaciji. Neprecizne, približne upute izražene riječima prirodnog jezika puno su primjenjivije u svakodnevnom realnom životu. Formalni prikaz približnih, nepreciznih vrijednosti, primjerice „*oko 100*", „*jako visoko*", „*vrlo sporo*", središnji je dio ***teorije lingvističke aproksimacije*** koja se izravno naslanja na ***neizrazitu teoriju skupova*** i ***neizrazitu logiku***. Već smo u dijelu o neizrazitoj logici spomenuli da su logika i teorija skupova usko povezane. Teoriju skupova možemo promatrati kao grafičku vizualizaciju logike, pa kako se Zadeh u prvim radovima o lingvističkoj aproksimaciji koristio neizrazitim skupovima, tako ćemo i mi u nastavku koristiti tu notaciju.

5.3.1 Lingvistička aproksimacija neizrazitim skupovima

Pojam ***neizrazitog skupa*** (engl. *Fuzzy Set*) nastao je poopćavanjem pojma klasičnog, standardnog skupa. Klasični skup (engl. *Crisp Set*) sadrži elemente (objekte) koji precizno zadovoljavaju uvjete zadanog skupa. Primjerice, skup brojeva *A* od 5 do 8 uključuje sve realne brojeve iz intervala [5,8]. Formalno ga definirano ***pridružnom*** (karakterističnom ili pripadajućom) ***funkcijom*** (engl. *Membership Function*) koja svakom elementu skupa pridružuje vrijednost 1 ako on ***pripada*** skupu, a vrijednost 0 ako ***ne pripada*** skupu. Označimo li pridruženu funkciju skupa *A* oznakom m_A, njena formalna definicija bi bila:

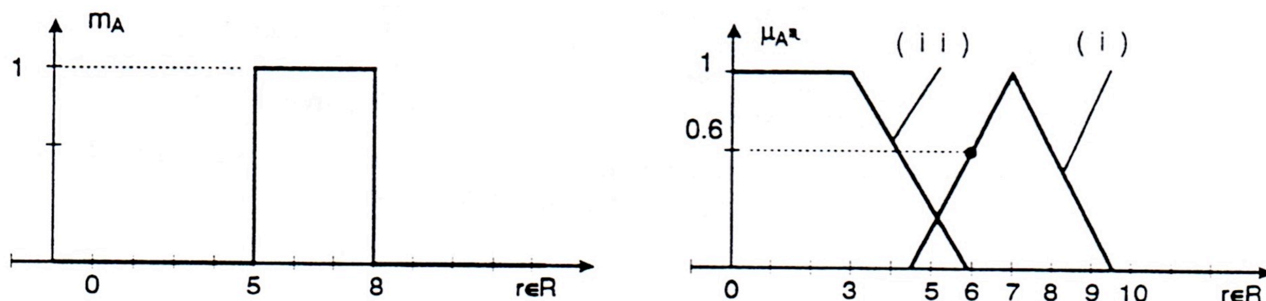
$$m_A: \Omega \rightarrow \{0,1\} \qquad (5\text{-}29)$$

gdje je Ω definicijski skup skupa *A*, skup na kojem je skup *A* definiran. Za prije spomenuti skup *A* realnih brojeva od 5 do 8, definicijski skup Ω je skup realnih brojeva ℝ pa skup *A* možemo formalno opisati pridruženom funkcijom:

$$m_A(r) = \begin{Bmatrix} 1 & za\ \ 5 \leq r \leq 8 \\ 0 & za\ ostale\ r \end{Bmatrix}$$

gdje je $r \in$ ℝ. Jezično ga interpretiramo: „*Skup A čine svi realni brojevi između 5 i 8, zato je vrijednost njihove pridružne funkcije jednaka 1".* Stavimo li na apscisu definicijski skup ℝ, a na ordinatu vrijednost pridružne funkcije m, skup A možemo prikazati crtežom na slici 5-16 lijevo.

Svaki realni broj *r* je „***ili u A ili nije u A***". Pridružna funkcija m_A pridružuje svim realnim brojevima *r* jednu od dvije moguće vrijednosti 0 ili 1, pa upravo zbog toga klasični skup odgovara klasičnoj dvovaljanoj logici: da ili ne, 1 ili 0, bijelo ili crno. U logici vrijednosti pridružne funkcije m_A zovu se vrijednosti istinitosti u odnosu na pitanje: „*Je li realni broj r u A ?*". Odgovor je DA, tvrdnja je **istinita**, onda i samo onda ako je $m_A(r) = 1$, u ostalim slučajevima kada je $m_A(r) = 0$ tvrdnja je lažna. „***Trećeg nema.***" jedan je od temeljnih postulata, kako klasične logike tako i klasične teorije skupova. On je i razlog zašto se klasičnim logičkim sustavom ne može opisati nesavršenost i nepreciznost realnog svijeta.



Slika 5-16. (Lijevo) – Pridružna funkcija običnog skupa „brojevi od 5 do 8". (Desno) – Pridružna funkcija neizrazitih skupova „brojevi oko 7" (i) i „brojevi oko 3 i manje" (ii)

Realni svijet nije crno-bijel, bio bi previše jednostavan da jest, realni je svijet siv, ponekad više bijel, ponekad više crn. Čovjek nije ili zločest ili dobar, dobrota svakog pojedinog čovjeka je negdje na ljestvici između ta dva ekstrema. Netko je samo „*dosta dobar*", netko je „*jako dobar*", a netko i „*jako, jako dobar*". Sve su to ***neizraziti skupovi,*** a Zadeh ih je formalno definirao na način da je proširio pojam pridružne funkcije i dozvolio da ona može poprimiti bilo koju vrijednost između ekstrema 0 i 1. Formalno izraženo ona može imati bilo koju vrijednost iz intervala [0;1]. Kako bismo istaknuli razlike, ovu novu pridružnu funkciju označit ćemo s μ, a neizraziti skup s A^* pa se pridružna funkcija neizrazitog skupa definira s:

$$\mu_{A^*} : \Omega \rightarrow [0,1] \qquad (5\text{-}30)$$

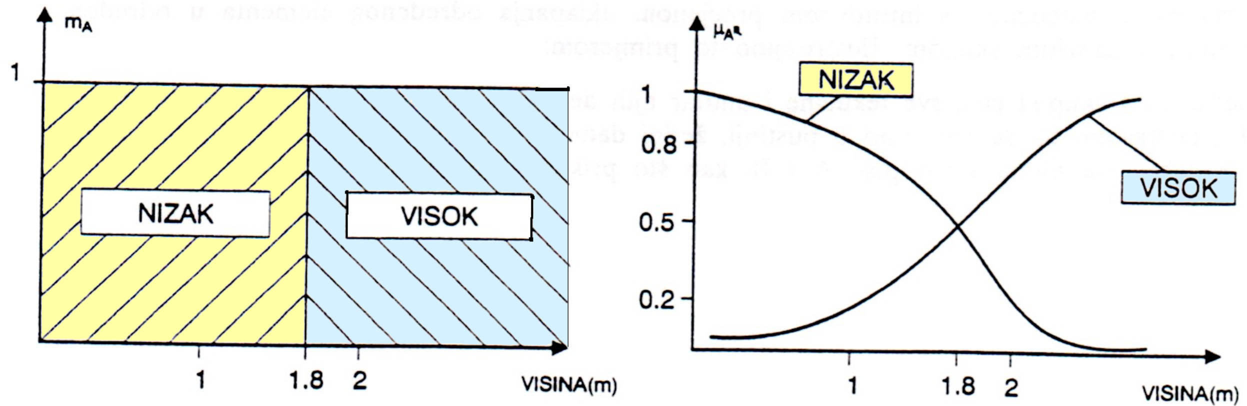
U ovom izrazu 0 i 1 su uključeni u moguće vrijednosti pridružne funkcije, pa je klasični skup samo poseban slučaj neizrazitog skupa. Vratimo se našem primjeru brojčanih vrijednosti te želimo prikazati vrijednosti „*oko 7*". Pojam „*oko 7*" nije precizan, on iskazuje nepreciznu vrijednost Ako ga definiramo klasičnim skupom i klasičnom pridružnom funkcijom (5-29), prikazujemo ga slikom 5-16 (lijevo). Prema njoj svi brojevi od 5 do 8 pripadaju pojmu „*oko 7*" s istom maksimalnom vrijednosti pridružne funkcije. Puno bliže našem intuitivnom shvaćanju pojma „*oko 7*" bila bi pridružnu funkciju prikazati krivuljom (*i*) sa slike 5-16 (desno). Prema njoj broj 7 sigurno pripada pojmu „*oko 7*", pa smo mu pridružili vrijednost pridružene funkcije 1, dok brojevima s lijeve i desne strane broja 7 pripadaju manje vrijednost pridružne funkcije i to što su dalje vrijednost pridružne funkcije je manja. Prikaz pojma „*oko 7*" krivuljom (*i*) sa slike 5-16 (desno) jedna je od mogućnosti definiranja tog pojma. Netko drugi bi kazao da treba proširiti granice, ili ih smanjiti, ili svim vrijednostima skupa ℝ dati neku vrijednost pridružene funkcije. Pojam „*oko 7*" nazivamo neizraziti skup i simbolički ga obično označavamo ga A^*, dodavši * kako bismo naglasili razliku između neizrazitog i običnog skupa. Izraz „*oko 7*" je njegov opis, njegova jezična vrijednost, a krivulja (*i*) sa slike 5-16 (desno) jedan od načina njegovog matematičkog definiranja mogućih prikaza. Za razliku od klasičnog skupa koji uvijek ima jedinstvenu pridruženu funkciju, svaki neizraziti skup može se prikazati s beskonačno mnogo različitih pridruženih funkcija. Definicija neizrazitog skupa ***nije jednoznačna***, ali ***nije ni proizvoljna***. Definicija pridružene funkcije treba se poklapati s našim intuitivnim shvaćanjem pojma koji se definira. Primjerice, sigurno nitko ne bi neizraziti skup „*oko 7*" prikazao pridruženom funkcijom (*ii*) sa slike 5-16 (desno). Ta bi pridružena funkcija više odgovarala neizrazitom skupu definiranom pojmom *brojevi „oko 3 i manji"*.

Iako možda na prvi pogled izgleda da je višeznačnost definiranja neizrazitog skupa slabost teorije neizrazitih skupova, to je u stvari njezina prednost. Žrtvovanjem jednoznačnosti dobije se fleksibilnost i mogućnost da se neizraziti skup maksimalno prilagodi stvarnoj situaciji i kontekstu u kojem se definira. Pojam „*velika temperatura*" sigurno će imati drugačije vrijednosti i granice ako se odnosi na temperaturu čovjeka ili temperaturu goruće zone rotacijske peći za proizvodnju cementa.

Definiranje pridružne funkcije neizrazitog skupa izrazom (5-30) omogućava postupni, kontinuirani prijelaz između pripadnosti i nepripadnosti određenom skupu. Posljedica takvog pristupa je da neizrazita logika nema samo dvije vrijednosti istinitosti, već beskonačno vrijednosti istinitosti o čemu smo već govorili u poglavlju 4.3.3. Odgovor na upit: „*Je li realni broj r u A ?*" sada glasi: „*Da, djelomično sa stupnjem pripadnosti* $\mu_{A^*}(r)$". Primjerice, ako je r = 6, a A^* definiran krivuljom (*i*) na slici 5-16 (desno), kažemo: „*Stupanj istinitosti da realni broj 6 pripada neizrazitom skupu 'oko 7' je 0,6.*"

Neizrazito zaključivanje i rasuđivanje temelji se na računanju s neizrazitim skupovima i neizrazitim relacijama. Kod definiranja neizrazitih skupova osim fleksibilnosti granica dozvoljava se i preklapanje granica. Primjerice, slika 5-17 prikazuje pridružene funkcije pojmova „*visok čovjek*" i „*nizak čovjek*" definiranih klasičnim skupom i neizrazitim skupom.

Važno je uočiti oštar prijelaz između niskih i visokih u slučaju klasične definicije. Svi ljudi ispod 1,8 m su niski, a svi ljudi iznad 1,8 m su visoki. Neizrazita definicija dozvoljava postupni prijelaz uz preklapanje. To znači da je čovjek visok 1,8 m „*visok*" sa stupnjem pripadnosti 0,5, ali u isto vrijeme i „*nizak*" sa stupnjem pripadnosti 0,5, on je negdje na sredini prijelaza između niskog i visokog. S druge strane čovjek visok 2 m pripada skupini visokih ljudi sa stupnjem pripadnosti 0,8, a skupini niskih ljudi sa stupnjem 0,2. Jasno je da ove definicije granica nisu apsolutne i da ovise o kontekstu zadatka.



Slika 5-17. Klasična (lijevo) i neizrazita (desno) definicija pojmova „visok" i „nizak"

U ovom primjeru imamo lingvističku varijablu *VISINA* koja ima svoje lingvističke vrijednosti „*visok*" i „*nizak*", ali ima i numeričke vrijednosti između 0,7 m i 3,0 m. Neizraziti skupovi prikazani na slici 5-17 (desno) daju značenje lingvističkim vrijednostima „*visok*" i „*nizak*" na način da se svakoj realnoj vrijednosti visine *x* pridružuje određena vrijednost pridružene funkcije. To se često formalno piše $\mu_{A^*}(x)$ ili jednostavnije *A* (x),* primjerice *VISOK(1,8) = 0,5* ili *NIZAK(2,0) = 0,2.*

U teorija neizrazitih skupova definirane su i različite operacije nad skupovima, od kojih su osnovne unija, presjek i komplementa. One u jezičnoj formi odgovaraju veznicima *I, ILI* i negaciji *NE*, a u neizrazitoj logici operatorima konjunkcije, disjunkcije i negacije. Pri tome se može koristiti bilo koji način definiranja ovih logičkih operatora *T* i *S* normama, o kojima smo govorili u dijelu poglavlja 4.3.3 posvećenom neizrazitoj logici, ali u praksi se najčešće koristi min-max logika definirana tablicom 4-5 koja za presjek koristi operaciju min, za uniju max, a za komplement operaciju minus:

$$A^*(x) \cup B^*(x) = max\,[A^*(x); B^*(x)] \qquad (5\text{-}31)$$

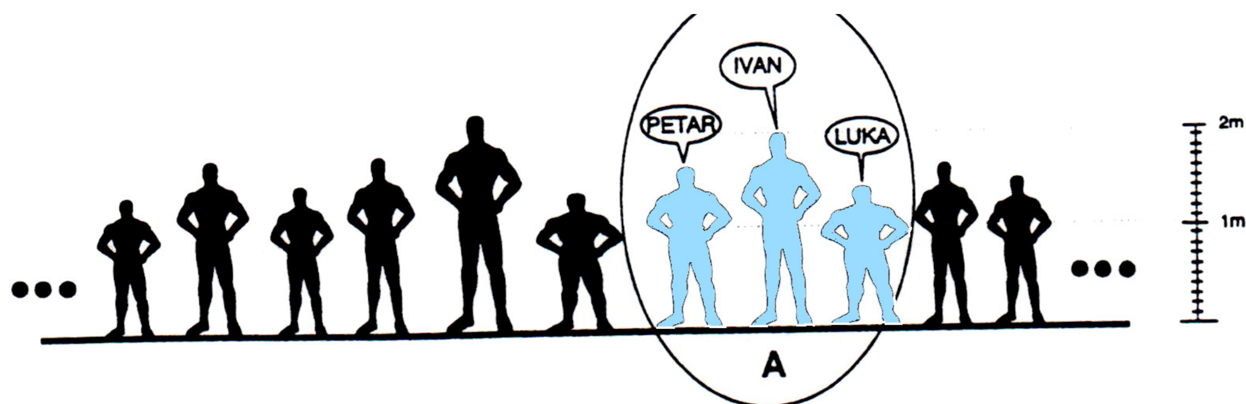
$$A^*(x) \cap B^*(x) = min\,[A^*(x); B^*(x)] \qquad (5\text{-}32)$$

$$\neg A^*(x) = 1 - A^*(x) \qquad (5\text{-}33)$$

Na primjer, vrijednost pridružene funkcije pojma *((VISINA 2 m JE VISOKA) i (VISINA 2 m JE NISKA))* je *min(0,8; 0,2) = 0,2.*

Zadeh i ostali istraživači nakon njega pokazali su i dokazali da je ovakav način definiranja operatora najbliži ljudskom intuitivnom shvaćanju tih pojmova u jezičnom obliku izraženih veznicima *I, ILI* i negacijom *NE.*

Svi do sada spomenuti neizraziti skupovi bili su **kontinuirani**, što znači da im je skup na kojem se definiraju bio kontinuum. Primjerice, pojmove *„veliki"* i *„mali"* definirali smo na intervalu realnih brojeva koji su predstavljali visine ljudi izražene u metrima. Skup svih realnih brojeva ℝ predstavljao je naš svijet rasuđivanja *Ω*, a unutar njega je definiran interval $H = [0,7\ ,\ 3] \subset \Omega$ koji sadrži realne vrijednosti visina ljudi. Skup *H* naziva se ***podrška*** (engl. *Support*) neizrazitih skupova *„veliki"* i *„mali".* Podrška neizrazitog skupa može biti i diskretan skup. U tom slučaju neizraziti skup ne definiramo krivuljom kao na slici 5-17, već diskretnim vrijednostima pridružene funkcije. Pogledajmo primjer. Neka je skup *Ω* skup svih ljudi i unutar njega izdvojimo podskup *A* koji čine samo tri čovjeka *A = {PETAR, IVAN, LUKA}* kao što prikazuje slika 5-18. Definiramo li na skupu *A* neizraziti skup *V* = „VISOKI LJUDI"*, tada njegova pridružena funkcija može na primjer glasiti *V* = 0,6/PETAR + 1/IVAN + 0/LUKA* gdje *0,6* predstavlja vrijednost pridružene funkcije kojom se procjenjuje s kojim stupnjem pripadnosti *Petar* pripada skupu visokih ljudi. Matematički to možemo označiti *V*(PETAR) = 0,6.*



Slika 5-18. Primjer definiranja diskretnog neizrazitog skupa čiju podršku čini skup troje ljudi: Petar, Ivan i Luka

U ovakvom načinu prikaza neizrazitog skupa + ne znači zbrajanje, već samo povezuje sve elemente pridružene funkcije neizrazitog skupa *V**. Umjesto takvog, dugog zapisa, češće se koristi kraći zapis tzv. ***neizrazitim vektorom*** *V* = [0,6; 1; 0]*. Kod ovakvog zapisa treba se precizno znati redoslijed elemenata podrške neizrazitog skupa *A = {PETAR, IVAN, LUKA}*. Kako bi se naglasila veza skupa podrške *A* i neizrazitog skupa *V**, piše se *Support(V*) = A* i čita: „*Podrška neizrazitog skupa V* je skup A.*"

Formalna definicija neizrazitog skupa kaže da je on skup uređenih parova, od kojih je prvi element podrške, a drugi njegova vrijednost pridružene funkcije. Za prethodni primjer to pišemo ovako:

$$V^* = \{(a\ , V^*(a))\},\ a \in P(V^*) = A \qquad (5\text{-}34)$$

gdje je *a* element podrške, a *V*(a)* njegova vrijednost pridružene funkcije. Neizraziti skup *V** u punom zapisu bi bio *V* = {(PETAR, 0,6), (IVAN, 1), (LUKA, 0)}*, ali se u praksi češće koristi neizraziti vektor kod neizrazitih skupova diskretne podrške ili prikaz krivuljom kao na slici 5-17 kod neizrazitih skupova kontinuirane podrške.

5.3.2 Lingvistička aproksimacija – neizrazite relacije

Sljedeći važan pojam je pojam ***neizrazite relacije*** (engl. *Fuzzy Relation*). Pretpostavimo da imamo dva dvočlana skupa ljudi: skup *X = {IVAN, PETAR}* i skup *Y = {JOSIP, LUKA}* i da želimo iskazati sličnost između svakog čovjeka iz skupa *X* i iz skupa *Y*. Svojstvo „*sličnosti*" je teško mjerljiva kategorija, ali lako procjenjiva. Pretpostavimo da je procjenitelju dozvoljeno da sličnost izražava koristeći skalu brojeva od 0 do 1. Što su ljudi sličniji, to im stupanj sličnosti treba biti bliže jedinici. Procjenitelj je svoju procjenu prikazao tablicom 5-6.

Tablica 5-12. Tablica s procjenom sličnosti skupa X = {IVAN, PETAR} i skupa Y = {JOSIP, LUKA}

	JOSIP	*LUKA*
IVAN	*0,8*	*0,6*
PETAR	*0,2*	*0,9*

Stupanj sličnosti Josipa i Ivana je *0,8*, a Luke i Ivana *0,6*. Najsličniji su Luka i Petar, a najmanje slični Josip i Petar. Tablica 5-6 nije ništa drugo nego pridružena funkcija neizrazite relacije koju smo nazvali *R* = „SLIČNOST"*. Neizrazita relacija je dvodimenzionalni neizraziti skup koji definiramo izrazom:

$$R^* = \{((x,y),\ R^*(x,y))\},\ (x,y) \in X \times Y \qquad (5\text{-}31)$$

Prvi dio uređenog para element je Kartezijevog skupa skupova *X* i *Y*, što drugim riječima znači element skupa kojeg čine sve moguće kombinacije skupova *X* i *Y*. U našem primjeru to su *X x Y = {(IVAN, JOSIP), (IVAN, LUKA), (PETAR, JOSIP), (PETAR, LUKA)}*. Drugi dio para su vrijednosti pridružene funkcije, u našem primjeru prikazane tablicom 5-6.

Kraći način zapisa neizrazite relacije je neizrazita matrica:

$$R^* = \begin{bmatrix} 0,8 & 0,6 \\ 0,2 & 0,9 \end{bmatrix}$$

ali se kod nje, kao kod neizrazitog vektora, treba znati redoslijed podrške. Definicijski izraz za R^* definira dvodimenzionalnu, binarnu neizrazitu relaciju, a općenito neizrazita relacija može biti i *n*-dimenzionalna.

Jedna od najvažnijih neizrazitih relacija je ***neizrazita relacija implikacije***. Ona je temelj ***neizrazitih produkcijskih sustava***. Jezični oblik neizrazite relacije implikacije je uzročno-posljedična rečenica:

„Ako je A iz X, onda je B* iz Y.”*

Npr.: „*Ako je **odstupanje** SREDNJE POZITIVNO, onda je **upravljanje** MALO NEGATIVNO*“.

ODSTUPANJE i *UPRAVLJANJE* su varijable, a *SREDNJE POZITIVNO* i *MALO NEGATIVO* njihove vrijednosti definirane odgovarajućim neizrazitim skupovima. Pogledajmo kako se uz pomoć neizrazitih pravila formira stručni (ekspertni) sustav temeljen na jezičnim pravilima i provodi neizrazito zaključivanje i rasuđivanje.

Neizraziti stručni (ekspertni) sustav temeljen na jezičnim pravilima je, kao što mu samo ime kaže, informacijski sustav koji omogućava korisniku dobivanje odgovarajućeg odgovora na pitanja iz određenog stručnog područja. Odgovori se dobivaju korištenjem stručnog znanja pohranjenog u bazi znanja koja se formira na temelju jezičnih pravila, te onda uz pomoć teorije lingvističke aproksimacije, temeljene na teoriji neizrazitih skupova, prebacuje u formalni matematički oblik. Ovakvi sustavi imaju mogućnost netrivijalnih odgovora i na pitanja koja nisu izravno pohranjena u bazi znanja, koristeći interpolaciju i interpretaciju pohranjenog znanja.

Kako je baza znanja stručnog sustava spremište ljudskog znanja koje je u svom temelju neprecizno, znanje je obično pohranjeno u obliku činjenica i pravila koja su često u obliku uzročno-posljedičnih jezičnih izjave oblika:

„Ako je UZROK, onda je POSLJEDICA.”

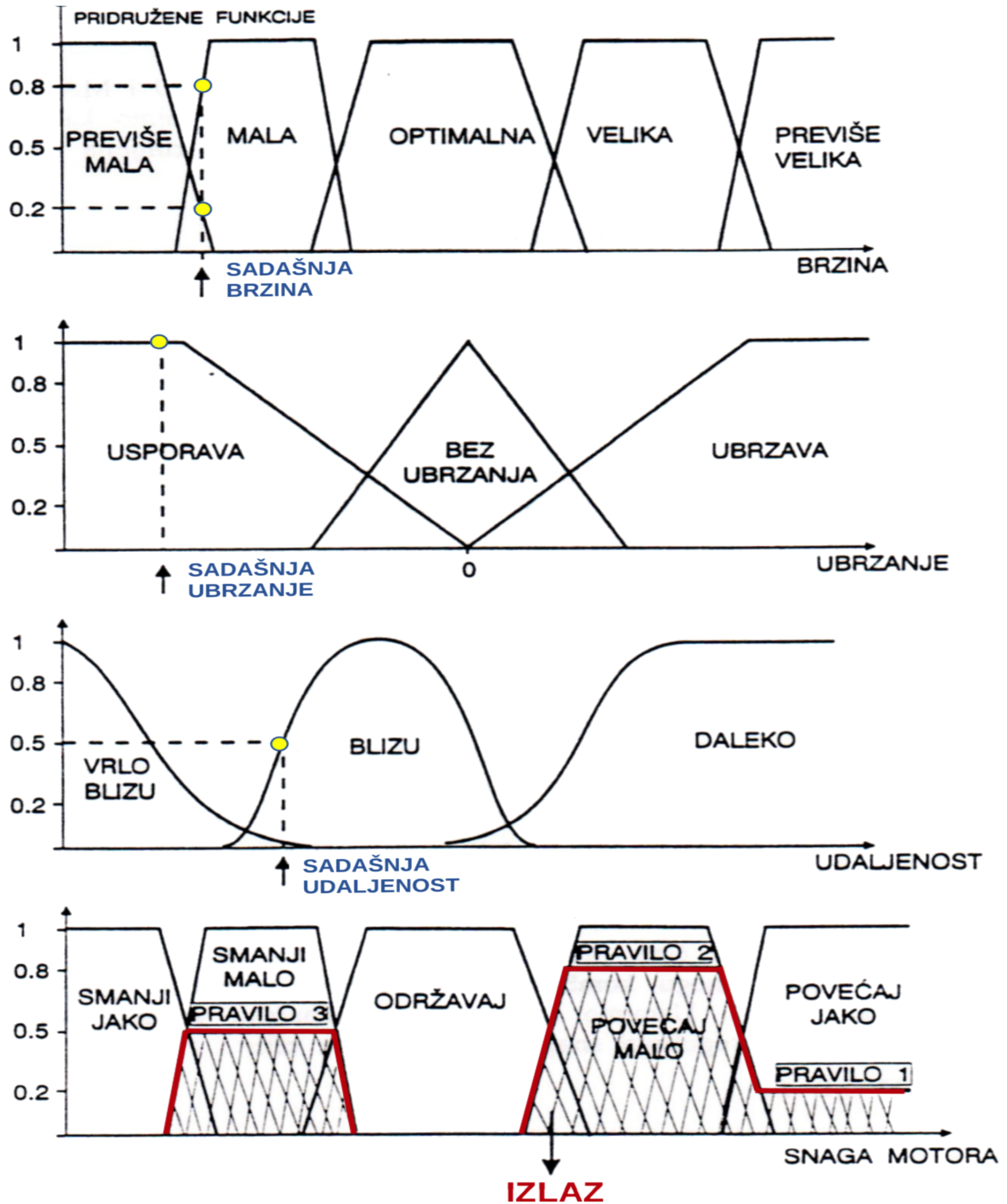
„Ako je SITUACIJA, onda je AKCIJA.”

„Ako je PRETPOSTAVKA, onda je ZAKLJUČAK.”

Već smo upoznali da se ovakav oblik iskazivanja znanja naziva ***produkcijsko pravilo***. Teorija neizrazitih skupova i neizrazita logika omogućile su ugradnju nepreciznosti, približnosti i realnosti u produkcijska pravila. Na taj su ih način približile čovjeku i njegovom stvarnom načinu opisa pojedine situacije ili akcije. Teorija neizrazitih skupova i neizrazita logika omogućile su da se produkcijska pravila neprecizno iskazana prirodnim jezikom prikažu odgovarajućim formalnim prikazom, te na taj način kodiraju za buduću računalnu primjenu. Jezične vrijednosti uzroka, posljedice, situacije i akcije modeliraju se odgovarajućim neizrazitim skupovima i kasnije koriste u postupku zaključivanja i rasuđivanja. Brojne su praktične primjene dokazale efikasnost i djelotvornost ovakvog pristupa.

5.3.3 Neizrazito zaključivanje

Osnovni dio neizrazitog stručnog sustava u kojem se provodi neizrazito zaključivanje čine jezična pravila, s tim da se i u uzročnom i u posljedičnom dijelu pravila varijable mogu kombinirati korištenjem veznika *i* (presjek neizrazitog skupa) i *ili* (unija neizrazitog skupa). Uzmimo za primjer automatsko vođenje vozila ilustrirano slikom 5-19.



Slika 5-19. Neizrazite vrijednosti ulazno-izlaznih veličina stručnog sustava za automatsko vođenje vozila, te primjer neizrazitog zaključivanja

Slika prikazuje na koji se način matematički definiraju osnovne lingvističke varijable *brzina, ubrzanje, udaljenost* i *snaga motora*. Ove varijable su povezane neizrazitim pravilima koja opisuju kako regulirati snagu motora budu:

PRAVILO 1. *– „Ako je **brzina** PREVIŠE MALA i **ubrzanje** USPORAVA, onda **snagu motora** POVEĆAJ JAKO."*

PRAVILO 2. *– „Ako je* ***brzina*** *MALA i* ***ubrzanje*** *USPORAVA, onda* ***snagu motora*** *POVEĆAJ MALO."*

PRAVILO 3. *– „Ako je* ***udaljenost*** *BLIZU, onda* ***snagu motora*** *SMANJI MALO."*

Pravila je odredio stručnjak (ekspert) koji dobro zna kako voziti automobil.

Pretpostavimo da se vozilo upravo uspinje uz brdo, da mu je brzina mala, da usporava i da je blizu vozila ispred sebe. Na Slici 5-19 označene su te sadašnje vrijednosti ulaznih veličina.

Pitanja su:

- *Kako na temelju neizrazitih pravila, definicije neizrazitih skupova, i sadašnjih stvarnih vrijednosti, donijeti zaključak i odrediti izlaz (potrebnu snagu motora)?*
- *Kako donijeti zaključak uzevši u obzir ova tri pravila?*

Neizrazito zaključivanje i rasuđivanje temelji se na ***kompozicijskom pravilu utjecaja*** (engl. *Compositional Rule of Inference*) koje se sastoji od tri koraka (slika 5-20):

1. određivanje stupanj zadovoljenja pravila
2. određivanje združenog neizrazitog skupa izlaza uzevši u obzir stupnjeve zadovoljenja pravila i
3. interpretiranje rezultata jednom jedinstvenom vrijednošću podrške izlazne varijable.



Slika 5-20. Shematski prikaz postupka neizrazitog zaključivanja

Proračunski dio kompozicijskog pravila utjecaja može se obaviti na više načina. Najjednostavnije i najčešće korišteno je tzv. ***max-min kompozicijsko pravilo utjecaja*** koje se na našem primjeru vođenja vozila sastoji u sljedećem:

Određivanje stupnja zadovoljenja pravila

Stupanj zadovoljenja pravila dobije se tako da se odredi vrijednost pridružene funkcije ulaznih varijabli za stvarne vrijednosti ulaznih varijabli, uz to da se veznik *i* interpretira kao presjek upotrebom operacije *min*:

u slučaju ***PRAVILA 1.*** to je:

PREVIŠE MALA (SADAŠNJA BRZINA) = 0,2

USPORAVA (SADAŠNJE UBRZANJE) = 1

STUPANJ ZADOVOLJENJA PRAVILA 1 = min (0,2; 1) = 0,2

za ***PRAVILO 2.*** vrijedi:

MALA (SADAŠNJA BRZINA) = 0,8

USPORAVA (SADAŠNJE UBRZANJE) = 1

STUPANJ ZADOVOLJENJA PRAVILA 2 = MIN (0,8; 1) = 0,8

za ***PRAVILO 3.*** vrijedi:

BLIZU (SADAŠNJA VRIJEDNOST) = 0,5

STUPANJ ZADOVOLJENJA PRAVILA 3 = 0,5

Očito je da će u postupku donošenja odluke najveći utjecaj imati ***PRAVILO 2***, ali i da utjecaji ***PRAVILA 3*** i ***PRAVILA 1*** neće biti zanemarivi.

Određivanje združenog neizrazitog skupa izlaza

Pojedini neizraziti skupovi izlaza formiraju se tako da se stupnjem zadovoljenja pravila ograniči neizraziti skup izlaza tog pravila. Slika 5-19 to prikazuje zorno. Na primjer: ***PRAVILO 3*** ima izlaz „*SMANJI MALO*". Njegov stupanj zadovoljenja je 0,5 pa iscrtkana površina prikazuje rezultat. Ovaj način proračuna je u stvari modeliranje neizrazite relacije implikacije operacijom minimum. Združeni neizraziti skup izlaza jednak je maksimumu svih pojedinih izlaza, što na Slici 5-19 prikazuje deblja linija ispod koje je crtkana površina.

Interpretiranje rezultata jedinstvenom vrijednošću

Kako nas najčešće zanima stvarna vrijednost izlazne varijable, u ovom slučaju snage motora, združeni neizraziti skup izlaza treba se prikazati jednom jedinom vrijednošću svoje podrške. Pri tome se najčešće koristi ***metoda težišta*** kojom se računa težište iscrtkanog združenog neizrazitog skupa. Na Slici 5-19 težište je označeno strelicom. Površine iscrtkanih dijelova lijevo i desno od težišta su jednake. Jezična interpretacija rezultata za naš primjer bi bila:

„*Povećaj* ***malo*** *snagu motora, zato što je vozilo ispred* ***blizu****, ali mi* ***usporavamo*** *zbog penjanja uz brdo.*"

ali nam to i nije važno. Važna nam je brojčana vrijednost snage motora.

Ovaj je postupak donošenja zaključaka detaljno prikazan na slici 5-20 jednostavan, ali vrlo efikasan. Najteži dio je, kao i kod svih ostalih stručnih sustava: Kako doći do pravog znanja – produkcijskih pravila? Kako to znanje izvući iz stručnjaka uzevši u obzir što on u stvari misli kada kaže: „*Brzina je previše mala.*" Ovim se poslom bave inženjeri znanja koji, osim tehničkog obrazovanja, imaju i obrazovanje iz područja psihologije i sociologije.

Primjer jednog stvarnog pravila vođenja rotacione peći za proizvodnju cementa je:

„*Ako je* ***temperatura zadnjeg dijela peći*** *JAKO SMANJENA i* ***postotak kisika*** *MALI*

i ***temperatura goruće zone peći*** *MALA, onda* ***brzinu peći*** *SMANJI i* ***dotok goriva*** *SMANJI.*"

Pri vođenju peći koristi se tridesetak ovakvih pravila.

Neizrazita logika je kao i svaka nova tehnologija od svog začetka 1965. godine do danas prolazila i faze euforije i faze razočaranja. Svaka nova tehnologija počinje naivnom euforijom. Početni entuzijazam često ljude potiče da očekuju više nego je moguće stvarno dobiti. Gotovo uvijek nagli rast očekivanja izazove protureakciju, najviše zbog toga što tehnologija nije još sazrela pa se obećanja ne mogu ispuniti. Velik broj ideja ovdje i završi i nikada više ne prođu prag pozitivnog očekivanja, bilo zbog toga što nisu bile prave ideje, bilo zbog toga što se prestalo u njih ulagati. One ideje koje prežive, prežive najviše zbog toga što je netko pronašao pravu korisnu primjenu. Iza toga slijedi novi poticaj razvoja i nove korisne primjene.

U razvoj neizrazite tehnologije uloženo je puno, što možda najbolje ilustrira podatak da je japansko ministarstvo trgovine (MITI) praćeno velikim proizvođačima, još 1988. godine uložilo u sedmogodišnji program komercijalne primjene neizrazite logike 150 milijuna američkih dolara. Središnji dio istraživanja odvijao se u novoosnovanom laboratoriju ***LIFE*** (engl. *Laboratory of Industrial Fuzzy Engineering*). Rezultati takvog ulaganja očituju se među ostalim i u tisućama patenata koje Japanci danas drže, a primjenjuju se od perilica za robu do podzemne željeznice. Dva tipična primjera uspješne primjene neizrazite logike su:

- Vođenje rotacione peći za proizvodnju cementa – sustav je u primjeni od 1982. godine i rezultirao je prosječnim smanjenjem potrošnje goriva za 2,5%, povećanjem čvrstoće cementa za 2,5% i produženjem vremena između dva remonta peći za 4 mjeseca.
- Predviđanje stanja na burzi namjenski rađeno za ***Yamachi Trading System***. Sastoji se od 800 neizrazitih pravila i prati dionice 65 industrija. Sustav je predvidio katastrofu na burzi (engl. *Black Monday*) osam dana prije.

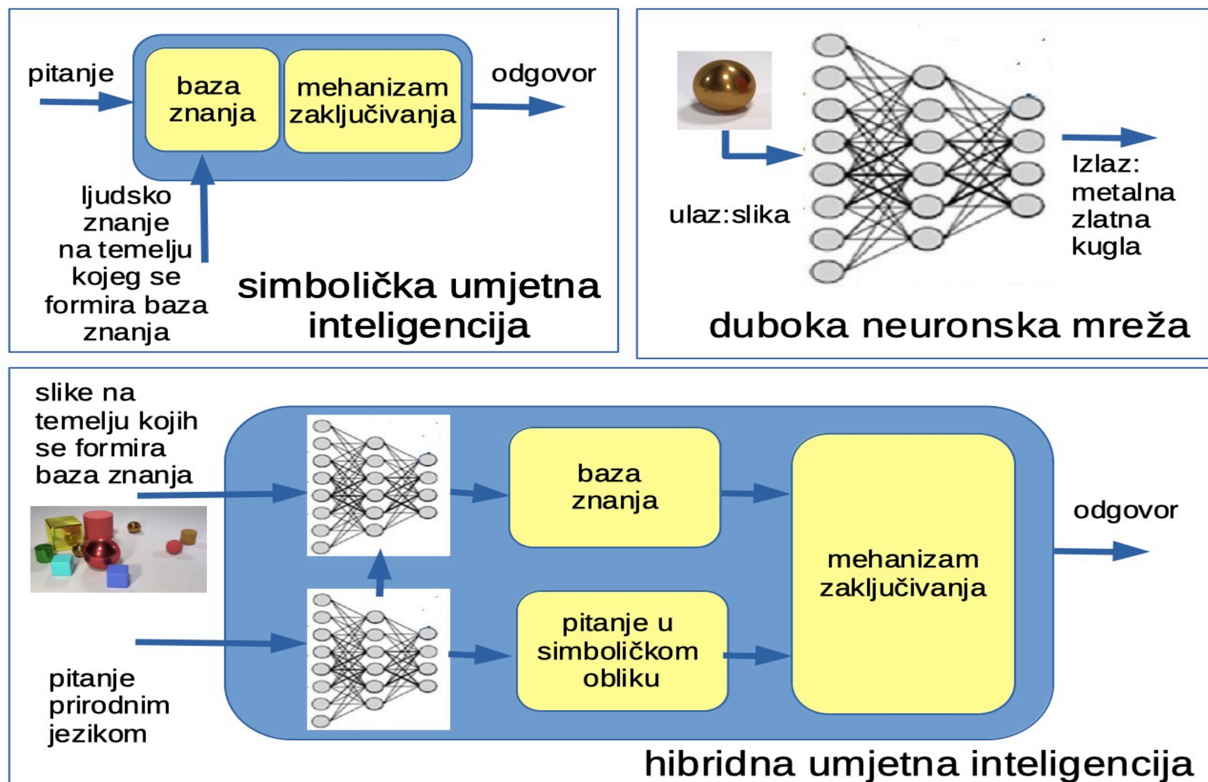
A sve je započelo 1965. godine u glavi profesora ***Lotfija A. Zadeha***. Njegov seminalni rad iz 1965. godine „*Fuzzy Sets*" (Zadeh, 1965) je najcitiraniji članak svih znanosti i svih vremena. Teorija neizrazitih skupova i neizrazita logika utjecale su na brojna područja ljudskog života što je dobro naglasio Zadehov učenik ***Bart Kosko*** u knjizi „*Fuzzy Thinking: The New Science of Fuzzy Logic"* (Kosko, 1993). Kako bi se istaklo značenje Zadehove teorije, poznata udruga inženjera *IEEE (Institute of Electrical and Electronic Engineers)* dodijelila je Zadehu 1995. godine nagradu za životno djelo (IEEE Medal of Honor).

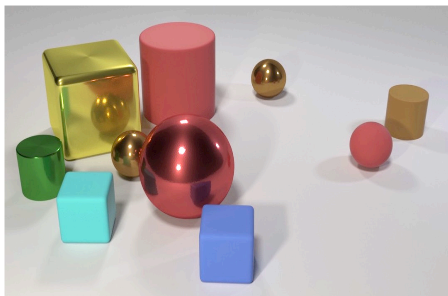
....

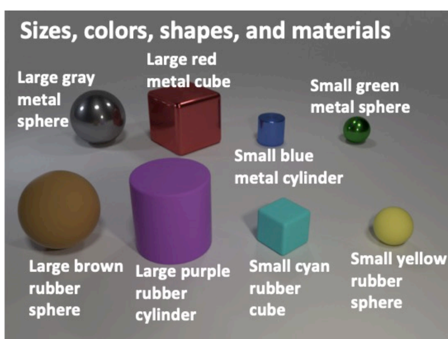
Dodatak: Posljednjih 10-ak godina ***duboko učenje*** (engl. *Deep Learning*) temeljeno na dubokim neuronskim mrežama, o kojem više govorimo u poglavlju 6.3.8., uspjelo je riješiti brojne probleme koje standardni postupci simboličke ili dobre stare umjetne inteligencije nisu uspjeli riješiti. Duboko učenje je posebno uspješno u zadacima prepoznavanja osjetilnih informacija (prepoznavanja objekata na slici, prepoznavanja teksta i slično). Iako se duboko učenje, usudili bismo se kazati ponekad i pomodno, nastoji samostalno upotrijebiti u rješavanju brojnih zadataka, posljednjih nekoliko godina velika se pažnja posvećuje integraciji dubokog učenja i dobro poznatih postupaka simboličke umjetne inteligencije. Rezultat ove fuzije je ***neurosimbolička umjetna inteligencija*** (engl. *Neurosymbolic AI*) ili ***hibridna umjetna inteligencija*** (engl. *Hybrid AI*) koja uspješno kombinira obje tehnologije. Slika ispod[86] shematski prikazuje način djelovanja simboličke umjetne inteligencije i dubokog učenja, te primjer njihovog združivanja u hibridnu umjetnu inteligenciju. Zadatak koji je ova mreža uspješno riješila je kompleksni zadatak ***vizualnog zaključivanja*** (engl. *Visual Reasoning*). 2016.g. istraživači sa ***Stanford University*** i ***Facbook AI Reserach*** objavili su rad vezan s vizualnim zaključivanjem o 3-D objektima na slici[87]. U ***CLEVR*** (*Compositional Language and Elementary Visual Reasoning*) bazi objavljeno je više od 100.000 računalno generiranih slika različitih objekata. Zadatak je bio napisati program koji bi uspješno izabirao slike na temelju verbalnih upita tipa: „*Postoji jednak broj velikih stvari i metalnih kugli?"* ili još složenije: „*Postoji kugla iste veličine kao metalna kocka; napravljen od istog materijala kao i mala crvena kugla?"*.

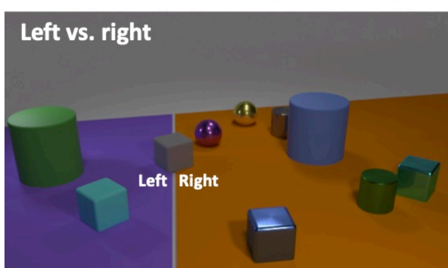
[86] Slika je prilagođena prema https://knowablemagazine.org/article/technology/2020/what-is-neurosymbolic-ai

[87] Slike i primjeri su iz rada *J.Johnson et al., CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning* - https://arxiv.org/pdf/1612.06890.pdf









Hibridni sustav umjetne inteligencije koristio je umjetnu neuronsku mrežu za prepoznavanje objekata na slici i formiranje baze znanja o njima, dok je simbolička ili dobra stara umjetna inteligencija, koristeći standardne postupke zaključivanja i formiranu bazu znanja, davala odgovore na postavljena pitanja. Mreža je bila uspješna u 92,6% slučajeva, pa je slična struktura kasnije korištena u rješavanju još složenijih zadataka kod kojih ulazni podatak nije bila slike nego video. Ova bi tehnologija mogla biti vrlo važna za autonomna vozila čije vrijeme dolazi.

Na slici lijevo[87] je primjer jedne od slika i lingvistički način definiranja različitih pojmova kojima se opisuju elementi slike, na primjer:

„Velika siva metalna kugla” (engl. *Large gray metal sphere)*

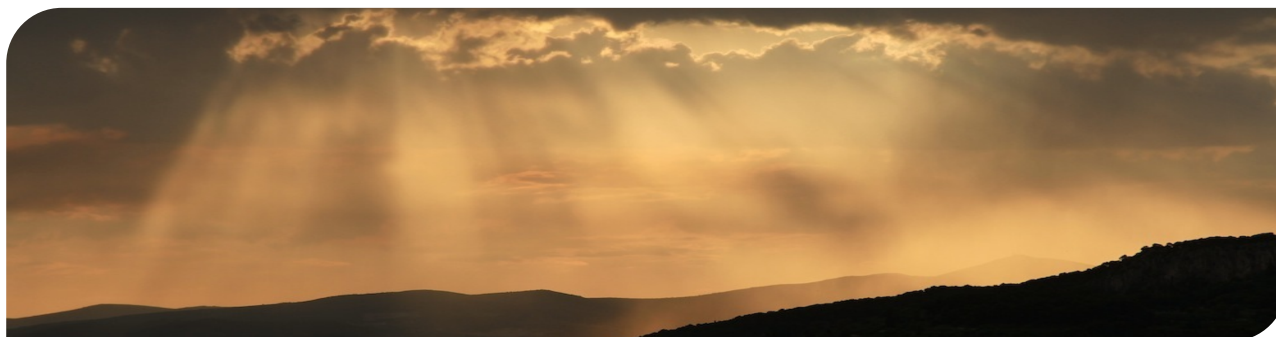
ili

„Mali metalni plavi valjak” (engl. *Small blue metal cilinder*)

ili

pojmovi *„lijevo“* (engl. *Left*) i *„desno“* (engl. *Right*).

6 UČENJE I STROJNO UČENJE





Strojno učenje danas je sigurno jedno od područja nebiološke inteligencije koje se najbrže razvija, posebno u odnosu na rješavanja brojnih kompleksnih zadataka koji se nisu mogli riješiti, bar ne na zadovoljavajući način. Sigurno je tome pridonio i razvoj računala koja su sve brža, sposobna napraviti brojne numeričke operacije na kojima se strojno učenje i zasniva. Nakon kratkog uvoda o učenju u ovom poglavlju dajemo pregled osnovnih postupaka strojnog učenja.

Prva asocijacija na riječ učenje kod većine je ljudi – školsko učenje. ***Školsko učenje*** ili ***edukacija*** bavi se pomaganjem prihvata znanja, vještina, ponašanja, vrijednosti… U učenju razlikujemo dvije uloge – ***učenika*** ili ***entitet koji prihvaća znanje*** i ***učitelja koji pomaže u postupku učenja***. Ipak, pojmom učenje možemo obuhvatiti puno širi spektar značenja, od nesvjesnog učenja jedinki životinjskog svijeta koje je bitno za preživljavanja do cjeloživotnog učenje ljudi kao stila života. U svim tim slučajevima učenje je prihvat novih znanja ili vještina.

Učenje se može odvijati svjesno i nesvjesno. ***Svjesno učenje*** događa se kada imamo namjeru usvojiti neko novo znanje ili vještinu, te svjesno i aktivno prikupljamo informacije koje vode prema tom cilju. Ovaj proces može biti potpomognut prisustvom učitelja koji upravlja cijelim ovim postupkom. ***Nesvjesno učenje*** događa se kroz interakciju s okolinom bez voljnog prikupljana i oblikovanja informacija. Nesvjesno učenje je prisutno i kod primitivnih jedinki.

Teoriju učenja u početku su izučavali filozofi. ***Platon*** (427. – 347. g. pr. n. e.) se prvi zapitao kako ljudi uče. Ovaj je filozof dao odgovor u vidu ***teorije rekolekcije*** ili ***Platonove epistemologije***. Prema njegovoj teoriji sve naučene informacije su zapravo prisjećanje prijašnjeg iskustva.

Britanski filozof ***John Locke*** (1632. – 1704.) ponudio je teoriju praznog stanja. On tvrdi kako se sva živa bića rađaju kao prazna ploča („*tabula rasa*"), bez ikakvog znanja, koju onda popunjavaju iskustvima.

6.1 Edukativna psihologija

Osim filozofije, učenjem se bavi i psihologija. Edukativna psihologija zaživjela je početkom 20. stoljeća, te predložila nekoliko teorija učenja.

6.1.1 Biheviorizam

Bihevioristički pristup učenju (engl. *Behaviorism*) utemeljio je ***John B. Watson*** (1878.-1958.). Njegove smjernice prate ***B. F. Skinner***, ***Ivan Pavlov*** i drugi. Bihevioristička teorija učenja jedna je od prvih teorija edukativne psihologije. Prema ovoj teoriji nova ponašanja ili promjene u ponašanju postižu

se formiranjem veza između stimulacija i odgovora. Učenik se promatra kao pasivan sudionik te se ne analiziraju njegovi interni procesi, već samo reakcije na podražaje. Učenje se postiže pozitivnim i negativnim potkrepljivanjem, tj. kaznama i nagradama kojima se forsiraju željene reakcije na podražaje. Glavna zamjerka biheviorizmu je ta da je učenik pasivan, a okolina aktivna. Također, kazna i nagrada nisu jedine motivacije učenja, već ljudi njeguju kompleksniji sustav vrijednosti koje ova teorija ne uzima u obzir.

6.1.2 Kognitivizam

Kognitivna teorija učenja (engl. *Cognitivism*) zaživjela je 50-ih godina 20. stoljeća, kao reakcija na biheviorizam koji ne promatra interne procese kod učenja. Ova teorija stavlja učenika u aktivnu ulogu i kaže kako je učenje zapravo obrada informacija koja vodi k razumijevanju i zadržavanju naučenog. Kognitivni konstruktivizam smatra da se nove kognitivne strukture aktivno kreiraju i nadograđuju na postojeće kognitivne strukture. Ovakvo objašnjenje procesa učenja vodi prema mogućnosti računalne implementacije modela učenja u obliku umjetne inteligencije koja uči. Zamjerke ovoj teoriji su što zanemaruje utjecaj okruženja (fizičkog, socijalnog, kulturnog). Također, učenje nije samo obrada informacija već konstruiranje sadržaja.

6.1.3 Konstruktivizam

Konstruktivizam (engl. *Constructivism*) se javlja 80-ih godina 20. stolčjeća, a temelji se na činjenici da učenje nije samo uspostava asocijativnih veza, kako tvrdi bihevioristička teorija, ili obrada informacija, kako tvrdi kognitivna teorija, već konstruiranje značenja i smisla. Ne naučimo samo ono što pročitamo i čujemo već najvećim dijelom ono što iskusimo. Učenje je proaktivan proces gdje učenik nove informacije interpretira i primjenjuje u vlastitu realnost.

6.1.4 Konektivizam

Konektivističku teoriju (engl. *Connectivism*) zovu još i teorija učenja za digitalno doba. Javlja se u 21. stoljeću i objašnjava kako Internet, te velika i jednostavna dostupnost informacija oblikuju nove paradigme učenja.

6.1.5 Računalna teorija učenja

Psihološke teorije učenje pokušavaju objasniti kroz ljudsko učenje. Od samih početaka računarskih znanosti znanstvenici pokušavaju stvoriti računalnu tvorevinu koja će imati približne mogućnosti kao i ljudske jedinke. Proces učenja pri tome nije iznimka. Računalna teorija učenja koja nam je ovdje i zanimljiva razvila se kao posebno područje umjetne inteligencije koje se bavi analizom i dizajnom algoritama ***strojnog učenja***.

6.2 Strojno učenje

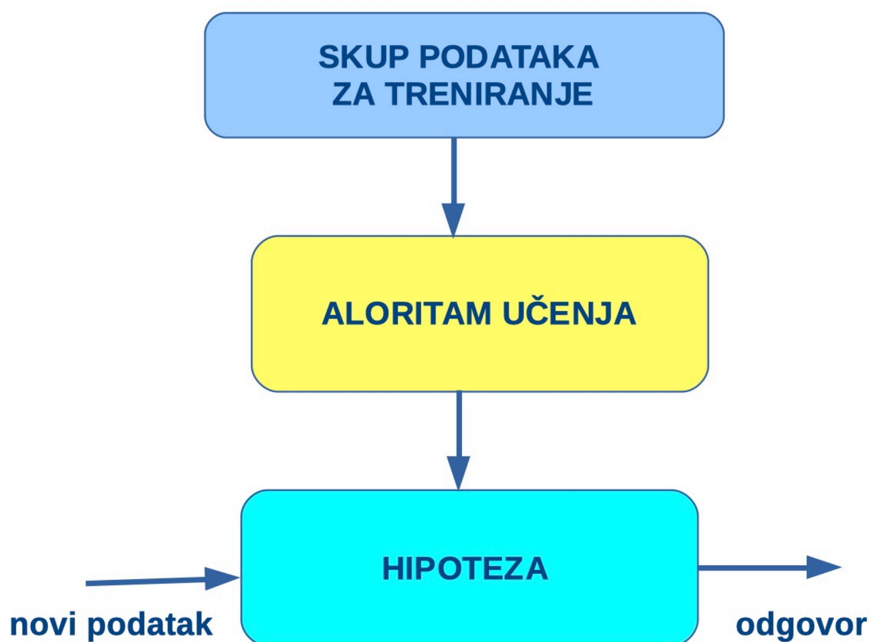
Strojno učenje (engl. *Machine Learning*) jedan je od pravaca umjetne inteligencije, te je ujedno i najzastupljeniji u posljednje vrijeme, kako u znanosti tako i u svakodnevnom korištenju. Ideja strojnog učenja jest postići da računala uče. Odavno znamo da su računala naprave koje lako pamte podatke. Razvojem tehnologije medija pohrane i distribuirane pohrane omogućili smo računalima da pamte podatke bez ograničenja količine, no pohrana velike količine podataka ne znači i usvajanje znanja ili vještine. U okviru strojnog učenja želja nam je da računala koriste te podatke za nova znanja i nova ponašanja. Strojno učenje bavi se idejom da računala sama stvaraju algoritme. U praksi, to se svodi na uočavanja zakonitosti iz podataka. Strojno učenje producira algoritme koji na temelju pohranjenih, dostupnih podataka generiraju hipoteze koristeći algoritme predviđanja. Postupak učenja iz podataka zove se ***treniranje algoritma strojnog učenja***.

Strojno učenje sadrži različite algoritme *„posuđene"* iz statistike, informacijske teorije, matematičke optimizacije, teorije automatskog vođenja, teorije grafova, fizike, kognitivne znanosti,

psihologije, objedinjene pod zajedničkim nazivom strojno učenje. Naziv je skovao ***Arthur Samuel*** (1901. – 1990.), jedan od pionira računalnih igara i umjetne inteligencije. On je 1959. godine dao i prvu definiciju: „*Strojno učenje je područje istraživanja u kojemu se računalu daje mogućnost učenja bez eksplicitnog programiranja.*" Arthur Samuel bavio se izradom programa za igru Dame (engl. *Checkers*) u kojoj program uči u interakciji s igračem.

Tom M. Mitchell (1951. –), američki znanstvenik, autor je knjige „*Machine Learning*" izdane 1998. godine. Njegova definicija strojnog učenja kaže: „*Kaže se da se računalni program uči iz iskustva E u odnosu na neki zadatak T s nekom mjerom uspješnosti P, ako se njegov učinak na T, mjereno s P, poboljšava s iskustvom E.*"

Algoritmi strojnog učenja imaju sposobnost uočavanja pravila (npr. algoritama i funkcija) iz skupa podataka kao što prikazuje Slika 6-1. Algoritam strojnog učenja za ulazne podatke uzima skup podataka za treniranje, a rezultat primjene algoritma strojnog učenja na ove podataka jest hipoteza. Hipoteza je uočena zakonitost koja daje metodu kako na temelju novih podataka odrediti odgovor. Cilj algoritma strojnog učenja je dati hipotezu koja se najbolje uklapa u skup podataka za treniranje. Što je više podataka za treniranje, to je i ishod učenja bolji. ***Andrew Ng***, profesor sa Stanforda koji predaje strojno učenje, je kazao: „*Ponekad ne pobjeđuje onaj koji ima najbolji algoritam, već onaj koji ima najviše podataka*".



Slika 6-1. Ilustracija postupka strojnog učenja

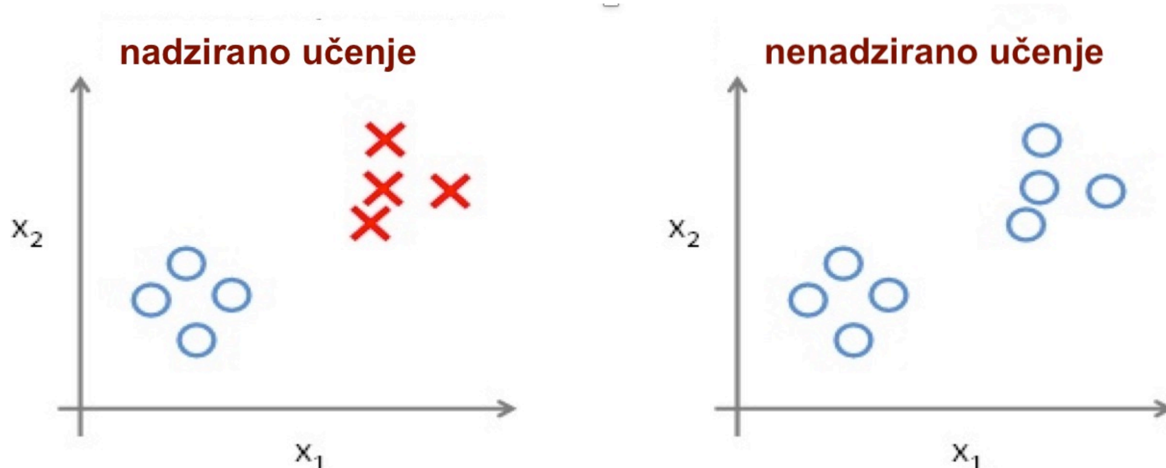
Primjer skupa podataka za treniranje mogao bi biti popis različitih bolesti i njihovih simptoma. Na taj bi se skup podataka mogao primijeniti neki od algoritama strojnog učenja da bi se dobila hipoteza. Hipotezi se tada može proslijediti skup novih simptoma i kao odgovor dobila bi se bolest koja ih uzrokuje.

Razlog zašto se znanstvenici u zadnje vrijeme intenzivno bave razvojem algoritama strojnog učenja razumljiv je. Velike količine podataka dostupne su na internetu, od generiranog sadržaja društvenih mreža, podataka dobivenih sa senzora ***Interneta stvari*** (engl. *IoT – Internet of Things*) i satelitskih informacija, do ***zapisa rada računanih sustava*** (engl. *Logs*), zapisa aplikacija, dnevnika rada itd. Svi ti podaci u sebi nose zanimljive informacije i znanja koje je potrebno otkriti, a ručno pregledavanje je zamorno i skupo.

Algoritme strojnog učenja dijelimo na vrste prema nekoliko različitih podjela. Prema ***načinu pripreme skupa podataka za obradu***, algoritmi strojnog učenja najčešće se dijele na:

- ***nadzirano učenje*** (engl. *Supervised Learning*) i
- ***nenadzirano učenje*** (engl. *Unsupervised Learning*).

Osnovna razlika između ova dva pristupa je u načinu pripreme skupa podataka za treniranje algoritma. Kod nadziranog učenja, algoritam radi s označenim podacima. Daje mu se skup značajki (npr. simptoma) i zaključaka (npr. bolesti koje uzrokuju te simptome). Na temelju tih zaključaka koje najčešće zovemo ***oznake*** (engl. *Labels*), algoritam nauči kako označiti nove značajke, a da se ta oznaka najbolje uklapa u uočene zakonitosti značajki ulaznog skupa. Kod nenadziranog učenja, podaci nad kojima se radi učenje su neoznačeni. Ulaz u algoritam je skup neoznačenih značajki, a zadatak algoritma je pronaći strukturu u tim podacima i donijeti vlastite zaključke. Iz ovog objašnjenja jasno je da je nenadzirano učenje mnogo kompleksniji problem od nadziranog.



Slika 6-2. Ilustracija razlike ulaznih podataka za nadzirano i nenadzirano učenje

Naravno da algoritmi nadziranog učenja imaju lakši posao, ali je u posljednje vrijeme dosta veći zahtjev za boljim i preciznijim algoritmima nenadziranog učenja. Lako dostupni podaci su uglavnom neoznačeni. Skup podataka senzora ili zapisa aplikacije lako je dobaviti, ali označavanje podataka je skup i zahtjevan posao. Kažemo da su „*podaci jeftini, ali su oznake skupe*".

Uz nadzirano i nenadzirano učenje, postoje još i hibridne tehnike ***polunadziranog učenja***, poput:

- ***pojačano učenje*** (engl. *Reinforcement Learning*) kod kojeg se učenje temelji na nagradi i kazni, te
- ***davanje preporuka*** (engl. *Recommender Systems*) kod kojega sustav istovremeno uči na primjer o proizvodima i kupcima te preporučuje kupcima proizvode koje preferiraju kupci slični njima.

6.3 Nadzirano učenje

Kod ***nadziranog strojnog učenja*** algoritmu učenja trebamo na početku pripremiti skup označenih podataka koji služi za treniranje algoritma. Označeni podaci koje zovemo ***oznake*** trebali bi sadržavati pouzdane parove ulazno-izlaznih informacija za koje znamo da su istinite. Na primjer, kod sustava medicinske dijagnostike to bi trebali biti parovi (simptomi, bolesti) na temelju kojih bismo istrenirali algoritam kako bi tijekom rada mogao zaključiti koja bi bolest mogla biti vezana uz pojavu određenih simptoma. Što je veći skup ulaznih podataka, to je algoritam bolje istreniran.

Postupcima nadziranog strojnog učenja možemo rješavati različite zadatke koje najčešće dijelimo u dvije grupe na temelju toga hoće li izlazna vrijednost koja se predviđa biti kontinuirana ili diskretna. Nazivamo ih zadacima:

- ***regresije*** (engl. *Regression*) koji predviđaju kontinuirane izlazne vrijednosti i
- ***klasifikacije*** (engl. *Classification*) koji predviđaju diskretne izlazne vrijednosti.

Kod regresije hipoteza daje postupak izračuna kontinuirane izlazne vrijednosti za neki novi ulazni podatak koji se najbolje uklapa u poznati skup podataka za treniranje. Kod klasifikacije hipoteza daje postupak izračuna kojoj je grupi iz podataka za treniranje najsličniji novi ulazni podatak.

Nadzirano strojno učenje primjenu nalazi u brojnim područjima, na primjer u već spomenutoj medicinskoj dijagnostici, predviđanju cijena proizvoda, predviđanju događanja, filtriranju neželjene elektroničke pošte, prepoznavanju znakova (engl. *Optical Character Recognition – OCR*) itd.

U nastavku ćemo prikazati nekoliko tipova regresije i klasifikacije, te načine njihovog rješavanja značajnim postupkom strojnog učenja koji se zove ***spust gradijenta***. Spust gradijenta je iterativni optimizacijski algoritam prvog reda kojim se traži minimum funkcije koja će se u našem slučaju nazivati ***funkcija cijene***. Predložio ga je još 1847. godine francuski matematičar ***Augustin-Louis Cauchy*** (1789. – 1857.), ali je značajnu primjenu dobio tek pojavom strojnog učenja koje se uvodi 60-ih godina 20. stoljeća.

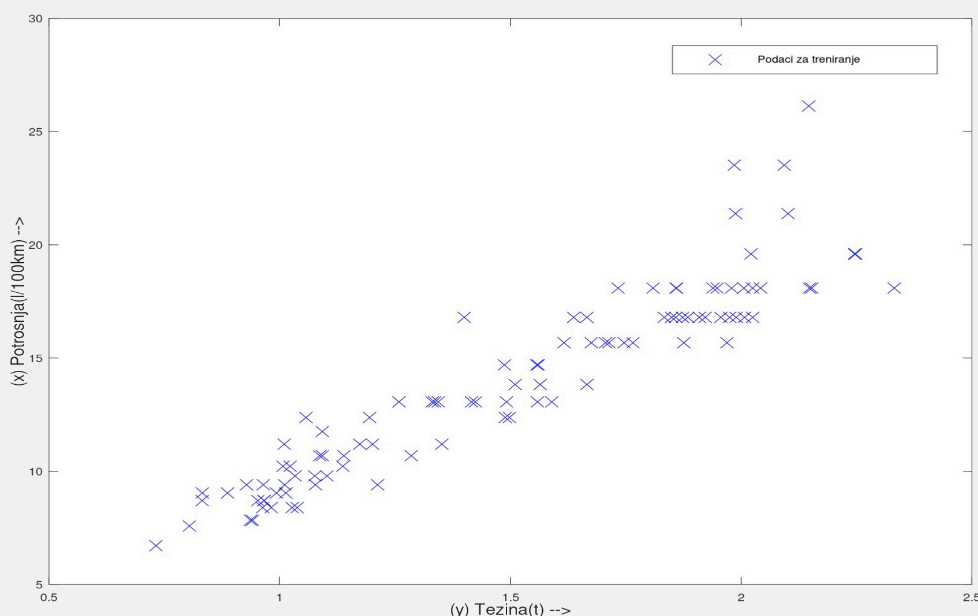
Posebno mjesto u nadziranom učenju imaju umjetne neuronske mreže koje u posljednje vrijeme doživljavaju veliki značaj u rješavanju brojnih zadataka, posebno klasifikacije.

6.3.1 Linearna regresija s jednom varijablom

Linearna regresija s jednom varijablom je zadatak predviđanja kontinuirane vrijednosti na temelju linearne funkcije uz promatranje samo jedne varijable. Pogledajmo najprije primjer s kojim ćemo je ilustrirati.

Zadatak 6. – Primjer linearne regresije

Linearna regresija između težine vozila i potrošnje goriva[88]. Ulazni podaci su 100 podataka prikazanih na slici 6-3 koji povezuju težinu vozila u tonama i potrošnju goriva u l/100 km.



Slika 6.3 – Potrošnja goriva u ovisnosti o težini vozila

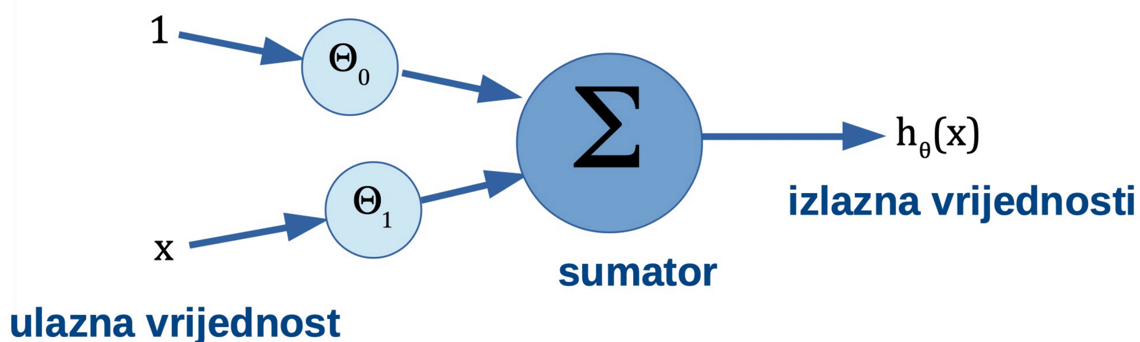
[88] Primjer i kod korišten kod ilustracije linearne regresije su prilagođeni podaci i kod prema https://github.com/adeveloperdiary/MachineLearning/tree/master/LinearRegression/Univariate_using_Octave

Na slici je moguće vrlo lako vizualno primijetiti trend potrošnje – što je automobil teži, to više troši. Osim rasipanja podataka pri kraju, mogli bismo i pretpostaviti da je ova ovisnost linearna. Cilj nam je odrediti tu ovisnost, kako bismo, ako nam je poznata težina vozila, lako procijenili kolika bi mu bila potrošnja.

Postupak je sljedeći: s x ćemo označiti varijablu o kojoj ovisi predviđanje (težina), a s y varijablu koju predviđamo (potrošnja). Funkciju predviđanja kojom izražavamo ovisnost cijene o godini proizvodnje zvat ćemo ***hipoteza***. Hipoteza je *„algoritam"* kojeg kao rezultat daje postupak strojnog učenja. Budući da je ideja napraviti linearno predviđanje, naša funkcija hipoteze imat će sljedeću formu:

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x \qquad (6\text{-}1)$$

Linearnu regresiju s jednom varijablom grafički možemo prikazati slikom 6-4.



Slika 6-4. Grafički prikaz linearne regresije s jednom varijablom

U formuli (6-1) θ_0 i θ_1 su parametri hipoteze. Cilj nam je parametre hipoteze podesiti tako da se funkcija hipoteze najbolje uklapa u podatke koje imamo. Nastojimo da za godinu proizvodnje automobila iz oglasa hipoteza predviđa vrijednosti cijene što bliže stvarnim cijenama iz oglasa. Zbog toga uvodimo ***funkciju cijene*** (engl. *Cost Function*). Funkcija cijene je funkcija parametara hipoteze (θ_0 i θ_1) i govori nam koliko je hipoteza kojoj damo određenu vrijednost parametara udaljena od stvarnih vrijednosti u skupu podataka. Vrijednost funkcije cijene za vrijednost parametara θ_0 i θ_1 možemo računati prema formuli (6-2):

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)})^2 \qquad (6\text{-}2)$$

Ova formula kaže da je vrijednost funkcije cijene za fiksne parametre jednaka sumi kvadrata udaljenosti vrijednosti hipoteze $h_0(x)$ i stvarnog podatka y iz skupa podataka, a m je broj podataka u skupu podataka. Cilj nam je pronaći takvu kombinaciju parametara θ_0 i θ_1 za koje će funkcija cijene biti minimalna. U našem primjeru, temeljem skupa podataka za treniranje, pronaći ćemo linearnu ovisnost godine proizvodnje automobila i cijene, takvu da za podatke iz skupa za treniranje ukupna udaljenost hipoteze i cijene bude minimalna.

Formalno napisano cilj postupka se iskazuje formulom (6-3):

$$\theta_0, \theta_1 \longrightarrow \min_{\theta_0, \theta_1} J(\theta_0, \theta_1) \qquad (6\text{-}3)$$

Ograničimo se sada (ilustracije radi) na samo jedan parametar θ_1, te zanemarimo θ_0. Pogledajmo kako se promjenom parametra θ_1 mijenja vrijednost funkcije cilja. Radi lakšeg praćenja koristit ćemo hipotetske podatke prikazane tablicom 6.1, a našem primjeru vratit ćemo se kasnije.

Tablica 6-1. Hipotetski podaci za ilustraciju traženja parametara linearne regresije

x	1	2	3
y	1	2	3

Tražimo hipotezu $h(x) = \theta_1 \cdot x$. Ako pretpostavimo da je $\theta_1 = 0$, hipoteza postaje: $h(x) = 0$.

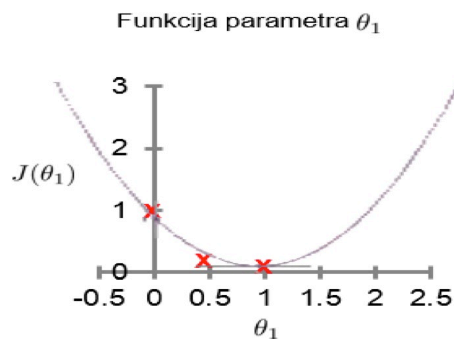
Tablica 6-2. Izlazna varijabla i hipoteza za $\theta_1 = 0$

x	1	2	3
y	1	2	3
h(X) za θ_1=0	0	0	0

Vrijednost funkcije cijene za $\theta_1 = 0$ je:

$$J(0) = \frac{1}{2 * 3}((1-0)^2 + (2-0)^2 + (3-0)^2) = \frac{14}{6} = 2{,}33$$

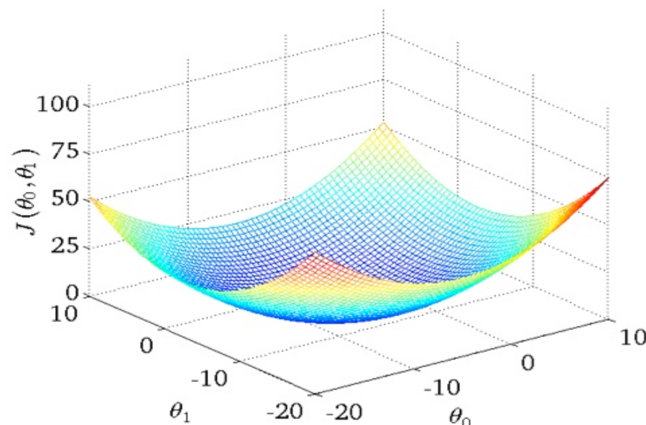
Ako pretpostavimo da je $\theta_1 = 1$, hipoteza postaje: $h(x) = 1 \cdot x$, a vrijednost funkcije cijene je $J(1) = 0$. Za $\theta_1 = 2$ hipoteza je $h(x) = 2 \cdot x$, a vrijednost funkcije cijene je $J(2) = 2{,}33$. Prikažemo li grafički funkciju cijene u ovisnosti o parametru θ_1, dobit ćemo sliku 6-5.



Slika 6-5. Grafički prikaz funkcije cijene u ovisnosti o parametru θ_1

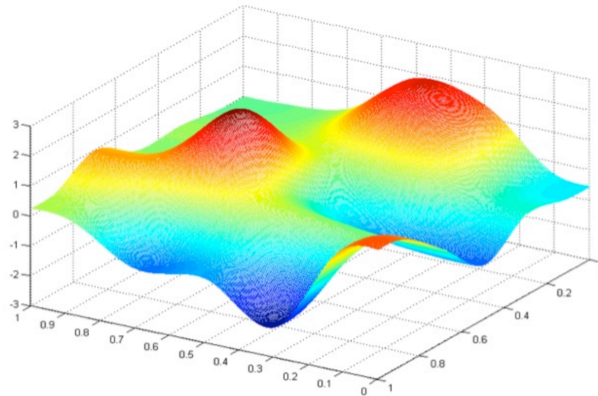
Slika 6-5 pokazuje da funkcija cijene ima minimum u vrijednosti $\theta_1 = 1$ i upravo je taj parametar najbolji kandidat za funkciju hipoteze u ovom jednostavnom primjeru.

Vratimo li se na hipotezu linearne funkcije s dva parametra, prikaz funkcije cilja u ovisnosti o dva parametra (θ_0 i θ_1) mogao bi izgledati kao na slici 6-6:



Slika 6-6. Funkcija cijene za dva parametra linearne hipoteze kod ujednačenih podataka

Najbolji kandidat za hipotezu jest kombinacija parametara na *„dnu"* površine ovakvog prikaza. Problem je što vrijednost funkcije cijene neće uvijek biti ovako pravilna. Češće će prikaz funkcije cijene izgledati puno složenije, s većim brojem minimuma i maksimuma, kao na primjer na slici 6-7.



Slika 6-7. Primjer prikaza funkcije cijene za dva parametra hipoteze

I u ovom slučaju bilo bi potrebno odabrati točku s kombinacijom parametara gdje je vrijednost funkcije cijene minimalna (najniža). Osim globalnog minimuma, ovakva funkcija ima i nekoliko lokalnih minimuma pa je pronalazak minimuma još više otežan.

Kod jednostavnog oblika funkcije cijene sa slike 6-6 zadatak linearne regresije možemo riješiti i analitički, na primjer metodom ***najmanjeg kvadratnog odstupanja*** (engl. *OLS – Ordinary Least Squares*). Kod funkcije cijene prikazane slikom 6-7, teško je analitički odrediti minimum vrijednosti funkcije, pa se zbog toga koriste iterativni postupci od kojih je posebno važan spust gradijenta.

6.3.2 Spust gradijenta

Spust gradijenta (engl. *Gradient Descent*) je postupak pronalaska lokalnog minimuma funkcije kroz iterativni postupak kojim se *„kreće"* po funkciji u smjeru najvećeg pada. Smjer pada funkcije matematički se pronalazi pomoću derivacije funkcije. Postupak se sastoji od toga da se slučajno odabiru početne vrijednosti parametara θ_0 *i* θ_1 te se zatim iterativno mijenjaju u smjeru pada funkcije cijene. Formalno, spust gradijenta možemo opisati formulom 6-4:

ponovi do konvergencije {

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (6-4)$$

}

gdje je α **parametar brzine učenja** (engl. Learning Componet Rate), a derivacijski član $\partial J / \partial \theta_j$ je **gradijentna komponenta** (engl. Gradient Component).

Kod linearne regresije s jednom varijablom koja ima dva parametra θ_0 i θ_1 algoritam spusta gradijenta može se opisati formulom 6-5:

ponovi do konvergencije {

$$\theta_0 = \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)})$$

$$\theta_1 = \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)}) \cdot x^{(i)} \qquad (6-5)$$

}

Za parametre θ_0 *i* θ_1 odaberu se proizvoljne početne vrijednosti, koje se zatim iterativnim postupkom mijenjaju. Vrijednost parametra se u svakoj iteraciji ažurira u smjeru pada vrijednosti funkcije cijene. Izbor parametra brzine učenja α je posebno kritičan. Premala vrijednost parametra α usporit će konvergenciju jer će se u svakom koraku iteracije napraviti mali korak. S druge pak strane, prevelika vrijednost parametra α može rezultirati prevelikim koracima te se optimalna vrijednost parametara θ_0 *i* θ_1 može preskočiti. Parametar m u nazivniku formule 6-5 predstavlja ukupan broj podataka skupa za treniranje.

Suma se računa tako da se za svaki podatak iz skupa za treniranje izračuna razlika hipoteze za trenutne parametre θ_0 i θ_1, oduzme se stvarna vrijednost podatka te se sumiraju svi podaci. Pseudokod ovog postupka bio bi sljedeći:

Algoritam 6-1. Pseudokod postupka za spust gradijenta

```
// inicijalizacija na početne vrijednosti
theta0 = 0;
theta1 = 0;
// iteracija
for (numiter) {
    // inicijalizacija sume na nulu
    delta0 = 0; //suma za ažuriranje parametra theta0
    delta1 = 0; //suma za ažuriranje parametra theta1
    // za svaki podatak iz skupa za treniranje
    for (i = 1 to m){
            delta0 += (theta0 + theta1*x[i] – y[i]);
            delta1 += (theta0 + theta1*x[i] – y[i])*x[i]
            }
    // update parametara na nove vrijednosti
    theta0 = theta0 - alpha * (1/m) * delta0;
    theta1 = theta1 - alpha * (1/m) * delta1
    }
```

Tablični zapis podataka za treniranje pogodan je i za ***matrični prikaz***. Uvodimo vektore $\boldsymbol{x}$ i $\boldsymbol{y}$. Vektor $\boldsymbol{x}$ sadržavat će težine automobila, a vektor $\boldsymbol{y}$ potrošnju goriva.

Zbog elegantnijeg matričnog zapisa, vektoru $\boldsymbol{x}$ ćemo dodati još jedan stupac na početku u kojemu će biti samo jedinice. Ovako dobivenu matricu označit ćemo s $\boldsymbol{X}$.

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_m \end{bmatrix}$$

Tražene parametre također ćemo predstaviti pomoću vektora $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

Sada hipotezu možemo u matričnom obliku pisati:

$$\boldsymbol{h}_\theta(x) = \boldsymbol{X} \cdot \boldsymbol{\theta} \qquad (6\text{-}6)$$

Matričnim množenjem matrica $\boldsymbol{X}$ i $\boldsymbol{\theta}$ dobijemo matricu vrijednosti hipoteze za svaki redak skupa za treniranje. Oduzimanjem matrice $\boldsymbol{y}$ dobit ćemo vrijednosti cijene svakog podatka skupa za treniranje za trenutne vrijednosti parametara θ_0 i θ_1. Kada vektor transponiramo u matricu jednog retka i m stupaca

i pomnožimo je s vektorom $\boldsymbol{x}$, dobit ćemo vektor s vrijednostima derivacija za parametre θ_0 i θ_1, te novi vektor $\boldsymbol{\theta}$ postupkom spusta gradijenta u matričnoj formi računamo jednadžbom 6-7:

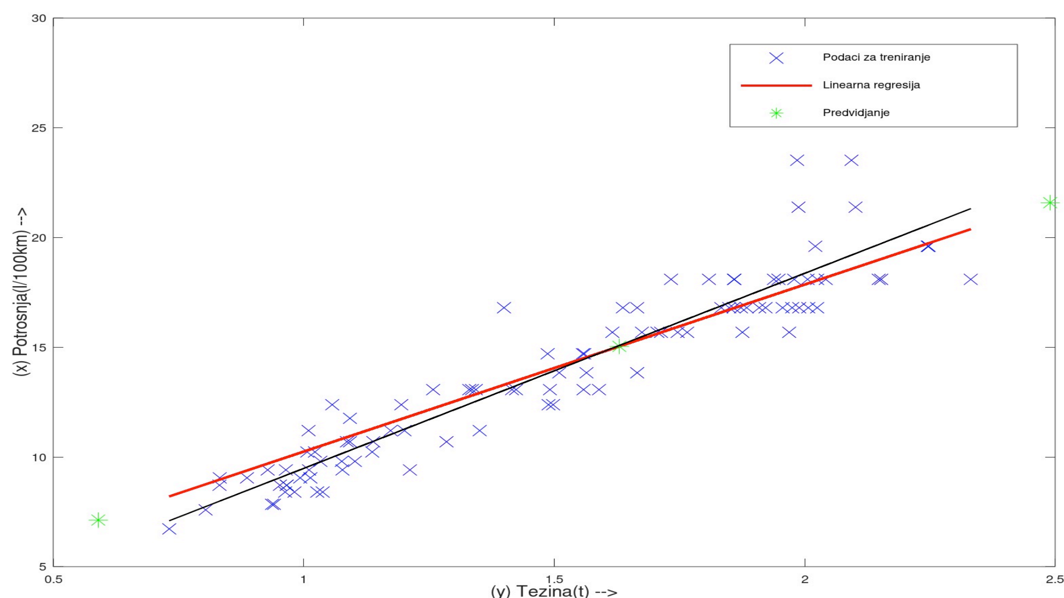
$$\boldsymbol{\theta} = \boldsymbol{\theta} - (\boldsymbol{\alpha}/\boldsymbol{m}) \cdot ((\boldsymbol{X} \cdot \boldsymbol{\theta} - \boldsymbol{y})^T \cdot \boldsymbol{x}) \tag{6-7}$$

Vratimo se sada na naš primjer i sliku 6-3 i primijenimo algoritam spusta gradijenta. Krećemo od nultih početnih vrijednosti $\theta_0 = \theta_1 = 0$. Tablica 6-3 prikazuje nekoliko prvih koraka za parametar brzine učenja $\alpha = 0,1$ koristeći ***Octave***[89]:

Tablica 6-3. Nekoliko prvih koraka traženja parametara linearne regresije metodom spusta gradijenta za podatke sa slike 6-3 za parametar brzine učenja α=0,1.

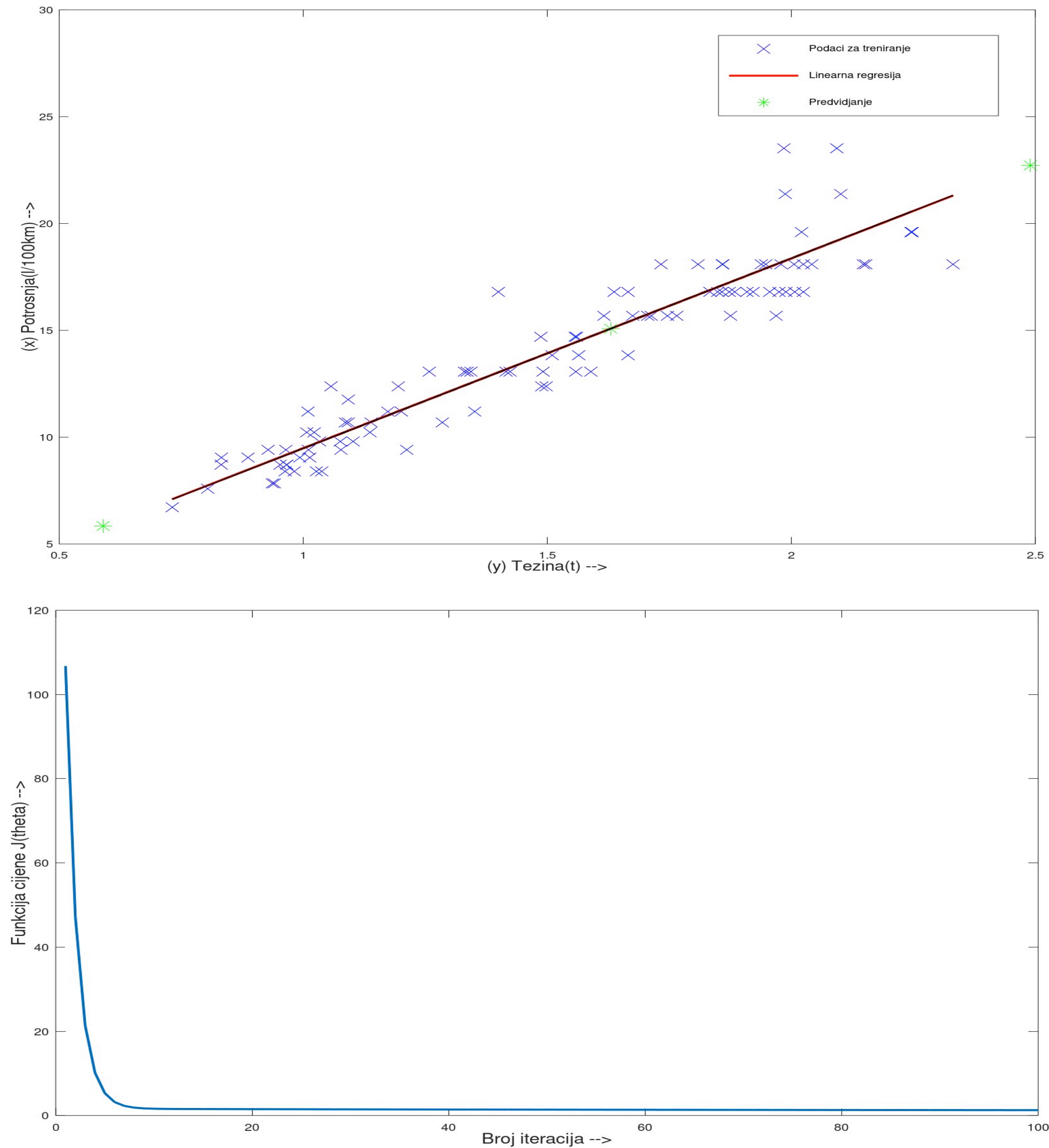
Korak	θ_0	θ_1	$J(\theta_0, \theta_1)$
1	1,400816	2,283847	106,752304
2	2,316807	3,792360	47,219340
3	2,913491	4,790153	21,387389
10	3,904009	6,674050	1,616149
100	2,630459	7,612992	1,274125
1000	0,600699	8,882528	1,110289

Analitička metoda najmanjeg kvadratnog odstupanja[90] daje točno analitičko rješenje $\theta_{0\text{-}ANALITICKI} = 0,587421$ i $\theta_{1\text{-}ANALITICKI} = 8,890934$ pa vidimo da se nakon 1000 iteracija spust gradijenta dosta približio analitičkom rješenju. Slika 6-8 prikazuje točnu jednadžbu linearne regresije (crno) i najbolju hipotezu (crveno) nakon 100 iteracija i nakon 1000 iteracija, te ovisnost funkcije cijene o broju iteracija.



[89] Octave je program za matematičke proračune https://www.gnu.org/software/octave/ .

[90] Korišten je *online* kalkulator https://planetcalc.com/5992/



Slika 6-8. Linearna regresija kod predviđanja potrošnje goriva za točno (analitičko) rješenje (crno) i rješenje dobiveno spustom gradijenta nakon 100 iteracija (gore) i 1000 iteracija (sredina) i ovisnost funkcije cijene o broju iteracija (dolje) za prvih 100 iteracija.

U ovom primjeru funkcija cijene u prvom dijelu brzo konvergira, ali zato u drugom dijelu konvergira sporo, pa veći broj iteracija i nema smisla. Na slici su ucrtane i tri zelene točke za predviđanje potrošnje goriva ako je težina vozila 0,59 tona, 1,63 tona i 2,49 tona.

Spust gradijenta opisan u ovom poglavlju je temeljni postupak. Razvojem strojnog učenja predložene su i njegove brojne modifikacije koje su u nekim elementima bolje, primjerice algoritmi *Momentum (1964.), AdaGrad (2011.), RMSprop (2102.), AdaDelta (2012.), Nesterov (2013.), Adam (2014.),*

AdaMax (2015.), Nadam (2015.), AMSGrad (2018.)[91]. Modifikacije se od osnovnog algoritma razlikuju uglavnom po modifikaciji parametra brzine učenja i/ili modifikaciji gradijentne komponente.

6.3.3 Linearna regresija s više varijabli

Svi znamo da cijene rabljenih automobila ne ovise samo i jedino o godini proizvodnje. Osim godine proizvodnje, cijeli je niz faktora koji utječu na konačnu cijenu automobila, poput proizvođača i modela, snage motora, broja prijeđenih kilometara itd. Faktore koji utječu na konačnu cijenu nazvat ćemo ***značajke*** (engl. *Features*). Broj značajki koje ćemo uzeti u obzir označit ćemo s n. Kod linearne regresije s jednom varijablom broj značajki koji je uzet u obzir bio je $n = 1$.

Ako predviđamo kontinuirane vrijednosti linearnom hipotezom temeljem više vrijednosti, radimo ***linearnu regresiju s više varijabli*** (engl. *MVLR – Multi Variable Linear Regresson* ili samo *Multiple Linear Regression*) kod koje hipoteza glasi:

$$h_{\boldsymbol{\theta}}(x_1, x_2, \dots, x_n) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \qquad (6\text{-}8)$$

Zadatak 7. – Primjer linearne regresije s više varijabli

Linearna regresija s više varijabli kod predviđanja cijene rabljenih automobila. Za potrebe primjera prikupljeni su podaci o cijenama rabljenih automobila s mrežne stranice za oglašavanje www.njuskalo.hr. Uzeti su oglasi u kojima su oglašavane prodaje rabljenih automobila te u kojima su navedene cijene i dodatne informacije o kojima će u nastavku biti više govora. Dio prikupljenih podataka za treniranje prikazan je u tablici 6-6. Imamo ukupno 6 različitih značajki ($n = 6$).

Tablične podatke također možemo prikazati matrično. Umjesto vektora $\boldsymbol{x}$ koji je sadržavao samo jednu varijablu sada imamo matricu s n stupaca, pa matrica $\boldsymbol{X}$ koja se kod linearne regresije s jednom varijablom sastojala od dva stupca – stupca jedinica i stupca s vrijednostima značajki, kod linearne regresije s više varijabli postaje matrica s $n+1$ stupcem – stupac jedinica i po jedan stupac za svaku od n značajki.

Neke značajke koje su nam bitne kod određivanja cijene nemaju numeričke vrijednosti, poput marke automobila ili boje. Kako se algoritam oslanja na numeričke proračune, potrebno ih je na neki način predstaviti (kodirati) brojevnim vrijednostima kako bi ih se moglo uzeti u obzir u proračunu. Tako ćemo npr. bijelu boju označiti brojem 1, sivu brojem 2, crnu brojem 3 itd. Sada nam boja kao značajka može biti navedena u matrici značajki $\boldsymbol{X}$.

Drugi način tretiranja nenumeričkih podataka s konačnim brojem nominalnih vrijednosti je „*one-hot encoding*” postupak. Za svaku vrijednost koju podatak može poprimiti uvodi se novi stupac kao nova značajka. Za svaki uzorak vrijednost samo jedne od grupe značajki je 1, i to za onu značajku koja odgovara vrijednosti, dok je za ostale vrijednost 0. Primjer ovakvog načina kodiranja za boju automobila za neke od automobila iz tablice 6-6 prikazan je u tablici 6-7.

Postupak proračuna parametara $\theta_0, \theta_1, \dots, \theta_n$ ostaje isti prema jednadžbi 6-4, odnosno u matričnoj formi jednadžbi 6-7. U našem primjeru pogledat ćemo što će linearna regresija dati ako nas zanima ovisnost cijene o godini (x_1), prijeđenim kilometrima (x_2) i broju vrata (x_3). Rezultat je[92]:

$$h_{\boldsymbol{\theta}}(x_1, x_2, \dots, x_n) = -11396000 + 5699 \cdot x_1 - 0{,}04865 \cdot x_2 + 6255 \cdot x_3$$

Zanimljivo je pogledati koje bi cijene automobila, za podatke koje smo koristili, dala ova jednadžba linearne regresije. Najveće odstupanje je naravno za podatke koji su najudaljeniji od pravca linearne regresije. Na primjer, za Mercedes E-klase 350 4MATIC cijena u oglasu je bila 144986 kn. Jednadžba linearne regresije s jednom varijablom daje 56032 kn, a jednadžba linearne regresije s

[91] Više detalja o ovim algoritmima može se pronaći npr. na https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9

[92] Korišten je on-line kalkulator https://home.ubalt.edu/ntsbarsh/Business-stat/otherapplets/MultRgression.htm

više varijabli daje 62840 kn. Taj je automobil imao mali broj prijeđenih kilometara, pa je cijena zbog toga i viša.

Tablica 6-6. Primjer podataka prodaje polovnih vozila

Marka	Motor	Prijeđeno km	Vrata	Godište	Boja	Cijena
VW POLO 1.4 TDI COMFORTLIN	1.4 TDI	264000	3	2003	žuta	28234
VW POLO 1.4 TDI COMFORTLIN	1.4 TDI	149000	5	2000	plava	15000
VW Polo 1,4	1.4 TDI	26100	5	2011	crvena	75545
VW Polo 1,9	1.9 TDI	288000	5	2003	bijela	26900
FORD FOCUS 1,6 TDCI	1.6 TDCI	104500	5	2010	crna	68601
Citroen Xsara 1,4 i	1,4 i	198000	5	2002	siva	22500
Renault Megane Grandtour	1,6 16V Sport	128000	5	2006	siva	49600
Opel Vectra 1,9 CDTI	1,9 CDTI	150000	5	2006	crna	57994
Mercedes E−klasa 350 4MATIC	?	94000	5	2006	crna	144986
Peugeot 207 1,4 Hdi	1,4 HDi	72000	3	2007	bijela	43114
Audi A3 1,9 TDI	1.9 TDI	220000	3	2008	bijela	76308
VW Polo 1,4 l	1.4 TDI	130000	3	2006	žuta	38564

Tablica 6-7. Primjer kodiranja boje automobila metodom „one-hot encoding"

Marka	bijela	crna	siva	crvena	plava	žuta
VW POLO 1.4 TDI COMFORTLIN	0	0	0	0	0	1
VW POLO 1.4 TDI COMFORTLIN	0	0	0	0	1	0
VW Polo 1,4	0	0	0	1	0	0
VW Polo 1,9	1	0	0	0	0	0

U oba ova slučaja pretpostavili smo linearnu ovisnost značajki, ali to u praksi nije uvijek slučaj. Ako sve značajke nisu linearno ovisne jedna o drugoj, treba primijeniti druge načine regresije, na primjer polinomalnu regresiju.

6.3.4 Polinomalna regresija

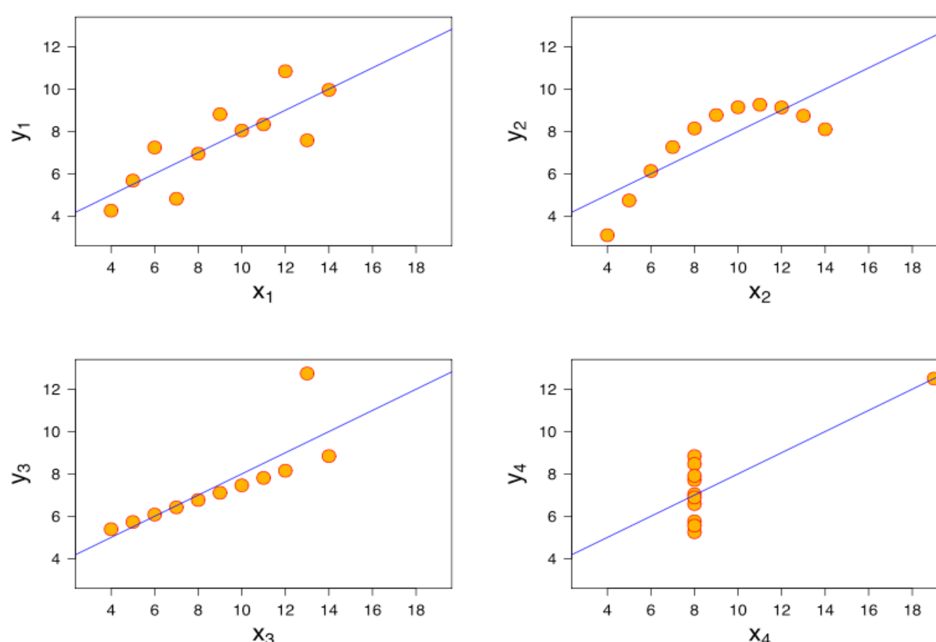
Polinomalna regresija (engl. *Polynomial Regression*) daje hipotezu u formi polinoma višeg stupnja (npr. 2., 3., 4. stupnja). Što je polinom većeg stupnja, to je funkcija hipoteze kompleksnija i moguće

je bolje približavanje funkcije hipoteze podacima iz skupa za treniranje. Na slici 6-9 prikazani su različiti skupovi podataka i njihova aproksimacija funkcijom linearne regresije.

Lako se vizualno može zaključiti da se za x_1, y_1 i x_3, y_3 može pretpostaviti da su linearno ovisne jedna o drugoj, ali za x_2, y_2, iako možemo temeljem podataka izračunati parametre linearne regresije, ta hipoteza nikad neće dobro raditi na novim podacima. Bilo bi bolje pretpostaviti neku polinomalnu hipotezu koja u pojedinim značajkama ovisi o zakonitostima koje su kompleksnije od linearnih. Dobro je što se prethodno opisani algoritam linearne regresije može lako prilagoditi i polinomalnoj regresiji jednostavnim proširenjem broja značajki i to za koliko smo pretpostavili da će biti stupanj polinoma.

Ako imamo skup značajki x_1 i x_2, i želimo donijeti hipotezu u formi polinoma drugog stupnja, tada ta proširena hipoteza glasi:

$$h_{\boldsymbol{\theta}}(x_{1,}x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_2 + \theta_4 x_2^2 \qquad (6\text{-}9)$$



Slika 6-9. Različiti skupovi podataka i njihova aproksimacija funkcijom linearne regresije

Matematički jednostavno proširujemo skup značajki na *x1, x12, x2 i x22*, pa nam matrica X sada ima stupac jedinica i još četiri stupca značajki, a *vektor* θ ima 5 redova. Daljnji postupak treniranja hipoteze i optimizacije parametara funkcije hipoteze postupkom spusta gradijenta ostaje identičan.

6.3.5 Pretreniranje i regularizacija

Što je stupanj polinoma veći, to je oblik funkcije hipoteze lakše uklopiti u dostupne podatke. Drugim riječima, kompleksnijom funkcijom hipoteze dobit ćemo hipotezu s manjom vrijednošću funkcije cijene. No ovisnosti su nekada jednostavnije nego što izgledaju, pa se može dogoditi da se korištenjem kompleksnije funkcije samo pretjerano zamaramo funkcijom cijene, dok bi jednostavnije hipoteze bolje opisale model pojave koju podaci opisuju.

Pretjerana pažnja na funkciju cijene izračunate na skupu podataka za treniranje može dovesti do tzv. ***pretreniranja*** (engl. *Overfitting*). Pretreniranje je pretjerano *„namještanje"* funkcije hipoteze da se što bolje uklopi u skup podataka za treniranje. Je li došlo do pretreniranja, možemo provjeriti na način da testiramo kako radi hipoteza za neke nove podatke koji se nisu nalazili u skupu podataka za treniranje. Kako nemamo uvijek dostupne nove podatke, u praksi se skup dostupnih podataka obično podijeli na podskup podataka za treniranje i podskup za testiranje, obično u omjeru 80:20. Hipoteza se trenira na

skupu za treniranje, te se provjeri vrijednost funkcije cijene na skupu podataka za testiranje. Ako hipoteza daje malu funkciju cijene za podatke za treniranje, a lošu (veliku) cijenu na skupu za testiranje, došlo je do pretreniranja. U tom slučaju potrebno je izabrati jednostavniju formu funkcije hipoteze, dovoljno jednostavnu da nema pretreniranja, a opet dovoljno složenu da dobro opisuje dani skup podataka. Princip ***Occamove britve***[93] (engl. *Occam's Razor*) kaže *„Ako dva rješenja daju iste rezultate, uvijek treba izabrati ono jednostavnije*". Konkretna primjena Occamove britve kod regresije naziva se ***regularizacija***. Regularizacija je uvođenje dodatnog izraza u funkciju cijene kojim se dodatno penalizira korištenje presložene forme funkcije hipoteze. Funkcija cijene dobiva sada oblik prikazan formulom 6-10:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_{\boldsymbol{\theta}}\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \cdot \sum_{j=1}^{n} \theta_j^2\right] \qquad (6\text{-}10)$$

Osim sume kvadrata udaljenosti hipoteze i stvarne vrijednosti, cijena se uvećava i za sumu kvadrata parametara hipoteze čime se postiže uvećanje cijene za veće vrijednosti parametra hipoteze. Dva su načina minimizacije ovakve funkcije cijene:

- smanjenjem udaljenosti vrijednosti hipoteze za dane podatke od stvarnih vrijednosti podešavanjem vrijednosti parametara na podatke iz skupa za treniranje, i
- smanjenjem ukupne sume kvadrata parametara hipoteze korištenjem jednostavnije funkcije hipoteze.

Minimizacija ovakve funkcije cijene postiže se balansiranjem između ova dva dijela – odabirom takvih parametara da je hipoteza dovoljno bliska podacima iz skupa za treniranje, a istovremeno i dovoljno jednostavna.

6.3.6 Klasifikacija logističkom regresijom

Logistička regresija (engl. *Logistic Regression*) je postupak klasifikacije korištenjem ideja linearne regresije. Problem klasifikacije se javlja kada imamo diskretne vrijednosti uzoraka koje smještamo u neku od klasa. Klasifikacija je predviđanje kojoj klasi uzorak pripada. Primjena ima puno, na primjer u klasifikaciji je li neka elektronička pošta smeće (engl. *Spam*) $(y=1)$ ili nije $(y=0)$, je li student na temelju rješavanja m broja zadataka prošao ispit $(y=1)$ ili nije $(y=0)$, je li neki tumor na temelju provedenih testova maligan $(y=1)$ ili nije $(y=0)$ itd.

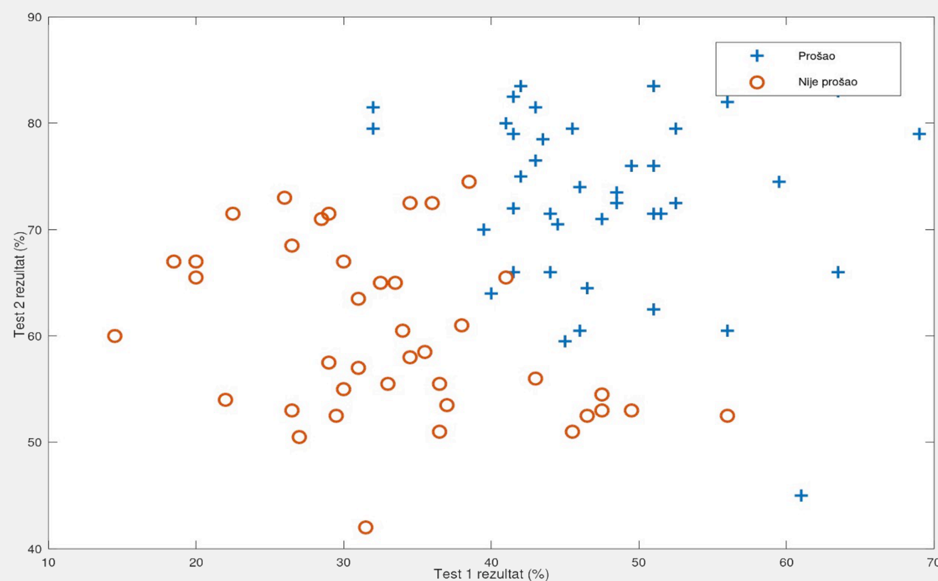
Zadatak 8. – Primjer linearne regresije s više varijabli

Za primjer ćemo uzeti predviđanje je li student prošao ispit na temelju rezultata dvaju testova (Test 1 i Test 2). Na slici 6-10 prikazani su prikupljeni podaci skupa za treniranje[94].

Svaki podatak predstavlja jednu točku određenu s koordinatama rezultata Testa 1 i Testa 2, a oblik oznake označava je li student prošao ispit (križić) ili nije (kružić).

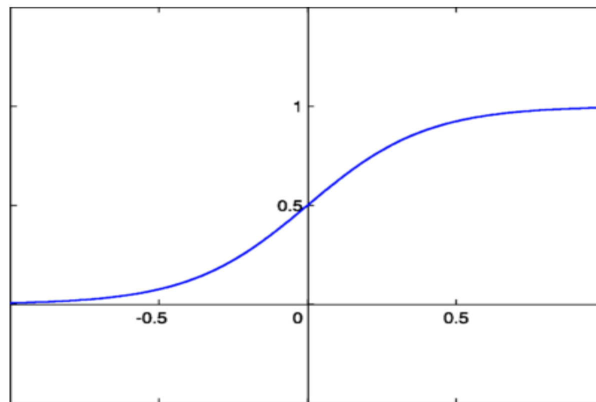
[93] Princip Occamove britve pripisuje se fratru ***Williamu Occamu*** (1287. – 1347.) i koristi se kao princip kod rješavanja problema. Ako više postupaka daje isti rezultat, onda uvijek treba uzeti najjednostavniji od njih.

[94] Primjer je dorađen prema ideji s http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex4/ex4.html



Slika 6-10. Ulazni podaci logističke regresije – na osima je postotak postignut na dva testa (Test 1 i Test 2), a oblik oznake kaže je li student prošao ispit ili nije

U ovom primjeru imamo dvije značajke, a u općem slučaju može ih biti m. Konačna odluka je binarna i nju donosi klasifikator na temelju ulaznih podataka. Kod logističke regresije hipoteza nije linearna kombinaciju značajki, već nelinearna funkcija linearne kombinacije značajki $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g(\boldsymbol{\theta}^T\boldsymbol{x})$, a kao funkcija najčešće se koristi Sigmoidna funkcija definirana jednadžbom (6-11) i prikazana slikom 6-11:



Slika 6-11. Sigmoidna funkcija

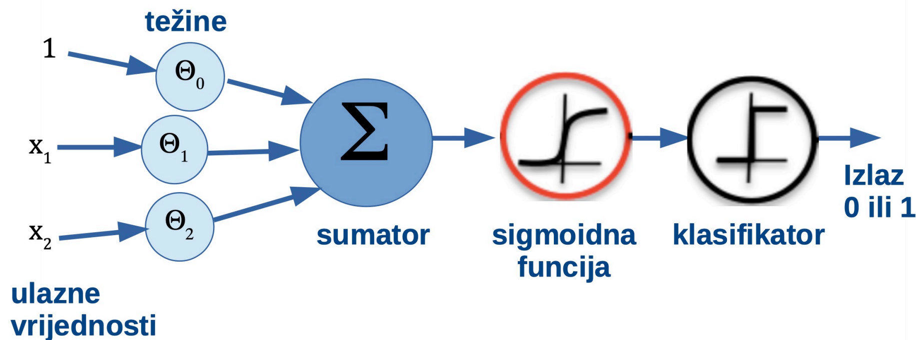
$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T\boldsymbol{x}}} \qquad (6\text{-}11)$$

gdje je $\boldsymbol{\theta}^T\boldsymbol{x}$ linearna kombinacija značajki:

$$\boldsymbol{\theta}^T\boldsymbol{x} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_m x_m = \theta_0 + \sum_{j=1}^{m} \theta_j x_j \qquad (6\text{-}12)$$

Sigmoidna funkcija daje vrijednosti od 0,5 do 1 za ulazne vrijednosti veće od 0, a vrijednosti između 0 i 0,5 za ulazne vrijednosti manje od 0. Važno je naglasiti da za velike ulazne vrijednosti sigmoidna funkcija ne daje znatno povećane izlazne vrijednosti kao linearna funkcija, već se izlazne vrijednosti asimptotski približavaju vrijednosti 1. Ako linearna kombinacija ulaznih značajki $\boldsymbol{\theta}^T\boldsymbol{x}$ rezultira vrijednostima većima od 0, sigmoidna funkcija daje vrijednost $h_\theta(x)$ veću od 0,5, a za ulazne

vrijednosti za koje je $\boldsymbol{\theta^T x}$ negativan (manji od 0) sigmoidna funkcija daje vrijednost manju od 0,5. Nama na izlazu treba binarna odluka pripadanja (1) ili nepripadanja (0), pa iza sigmoidne funkcije trebamo staviti klasifikator koji za $h_{\boldsymbol{\theta}}(x) \geq 0{,}5$ daje 1 (pripadanje klasi 1), a za vrijednosti $h_{\boldsymbol{\theta}}(x) < 0{,}5$ daje 0 (pripadanje klasi 0).Grafički prikaz logističke regresije u dvije klase na temelju dviju značajki prikazuje slika 6-12.

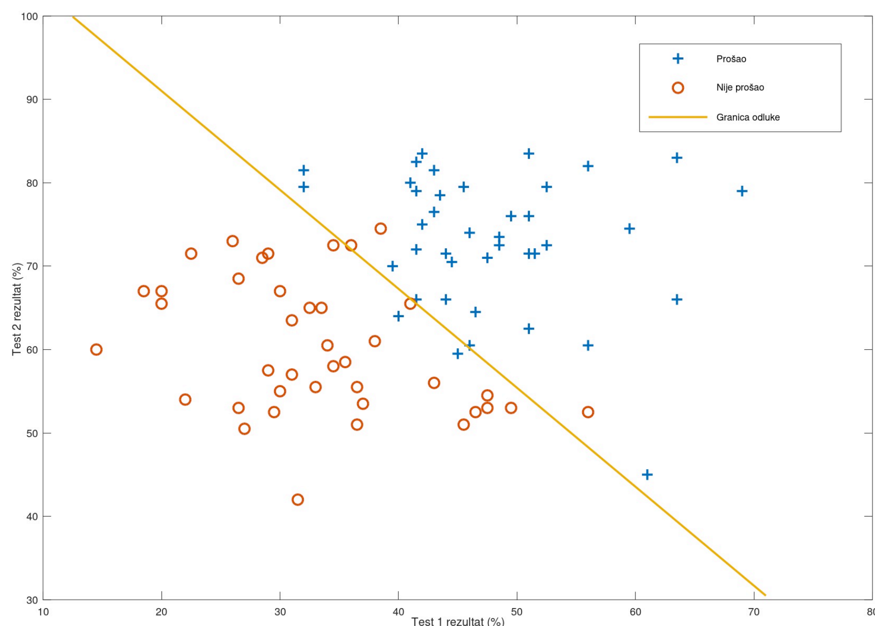


Slika 6-12. Grafički prikaz logističke regresije u dvije klase

Kako interpretirati ovako definiranu funkciju hipoteze? Uvodimo novi pojam koji se zove ***granica odluke*** (engl. *Decision Boundary*).

Granica odluke

Pogledajmo na primjer podatke s dvije značajke na temelju kojih ulazne podatke klasificiramo u dvije klase (slika 6-10). Pitanje je možemo li postaviti jasnu granicu između skupa uzoraka „*prošao*" i „*nije prošao*". Ako bismo granicu mogli predstaviti formulom jednadžbe pravca, tada bi nju definirao izraz $\boldsymbol{\theta^T x} = 0$, odnosno u slučaju kod kojeg imamo dvije značajke $\theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 = 0$. Kod sigmoidne funkcije kombinacije značajki x_1 i x_2 za koje je ova linearna kombinacija veća od 0 u koordinatnom sustavu prikazuju se točkama iznad pravca. Za takve vrijednosti hipoteza će predvidjeti pripadnost klasi 1. Kombinacije za koje linearna kombinacija ima vrijednosti manje od 0 nalaze se ispod pravca i hipoteza predviđa pripadanje klasi 0. Zbog toga se linija predstavljena izrazom $\boldsymbol{\theta^T x} = 0$ naziva ***granica odluke*** za logističku regresiju. Na Slici 6-13 prikazana je granica odluke za primjer prolaznosti ispita sa Slike 6-10 dobivena programskom bibliotekom ***Octave***.



Slika 6-13. Granica odluke za ulazne podatke prolaznosti na temelju rezultata dvaju testova.

Pitanje je kako smo došli do ove jednadžbe pravca. Osnovni je problem, kao i u prethodnim slučajevima, odrediti vektor težina $\boldsymbol{\theta}$. Na svu sreću to je ovdje moguće napraviti algoritmom spusta gradijenta uz određene modifikacije funkcije cijene.

Funkcija cijene

Funkcija cijene hipoteze logističke regresije mora biti izražena drugačije nego kod linearne regresije. Greška predviđanja pojedinog podatka je 0 ili 1, ovisno o tome je li ispravno ili neispravno hipotezom predviđeno pripadanje klasi 1 i klasi 0.

Za jedan podatak matematički izraz za funkciju cijene logističke regresije koju smo označili funkcijom *Cost()* je:

$$Cost(h_{\boldsymbol{\theta}}(x), y) = \begin{cases} -log\big(h_\theta(x)\big), & za\ y = 1 \\ -log\big(1 - h_\theta(x)\big), & za\ y = 0 \end{cases} \qquad (6\text{-}13)$$

Kada sumiramo po svim podacima, dobije se funkcija cijene hipoteze za određenu kombinaciju parametara $\boldsymbol{\theta}$:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \cdot Cost_1\big(h_{\boldsymbol{\theta}}(x^{(i)}), y^{(i)}\big) + \big(1 - y^{(i)}\big) \cdot Cost_1\big(h_{\boldsymbol{\theta}}(x^{(i)}), y^{(i)}\big)\right] =$$

$$= -\frac{1}{m}\left[\sum_{i=1}^{m}(y^{(i)}\ log\ h_{\boldsymbol{\theta}}\big(x^{(i)}\big) + (1 - y^{(i)})log\ (1 - h_{\boldsymbol{\theta}}(x^{(i)}))\right] \qquad (6\text{-}14)$$

Ovo možemo interpretirati na način da želimo maksimizirati vjerojatnost da se slučajno odabrani uzorak pravilno klasificira, što se uobičajeno naziva ***maksimalna procjena vjerojatnosti*** (engl. *Maximum Likelihood Estimation*). Na ovu funkciju cijene možemo primijeniti i regularizaciju (poglavlje 6.3.5).

Spust gradijenta za logističku regresiju

Algoritam za spust gradijenta kod ovako postavljene funkcije cijene ostaje isti kao i za linearnu regresiju. Tražimo $min_{\boldsymbol{\theta}}\ J(\boldsymbol{\theta})$ na način da se u svakoj iteraciji parametri mijenjaju u smjeru pada funkcije cijene:

ponovi do konvergencije {

$$\theta_j = \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)}) \cdot x^{(i)} \qquad (6-15)$$

}

što je u biti ista jednadžba kao kod spusta gradijenta linearne regresije.

U našem primjeru sa slike 6-12 konačne vrijednosti vektora težina $\boldsymbol{\theta}$ su θ_0=*-44,26458*, θ_1=*0,45777*, i θ_2=*0,38577*.

Logistička regresija u više klasa

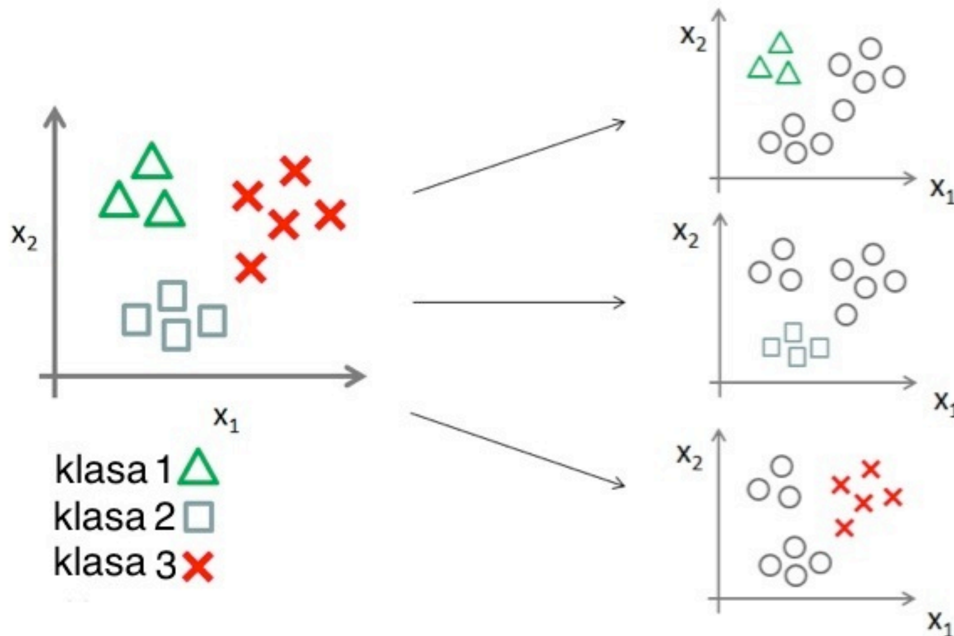
Opisani algoritam logističke regresije koristan je za tzv. ***binarnu klasifikaciju*** kod koje radimo klasifikaciju u dvije klase, pa izlaz hipoteze uzima vrijednosti iz dvočlanog skupa *{0,1}*. Ako podatke trebamo smjestiti u više klasa, algoritam se treba tome prilagoditi. Jedan od načina prilagodbe je metoda „***jedan protiv svih***" (engl. *One vs. All* ili *One vs. Rest*) koju prikazuje slika 6-13. Ova se metoda temelji na treniranju više klasifikatora, po jedan za svaku klasu.

Npr. ako imamo 3 klase, postupak logističke regresije provest ćemo 3 puta:

- u prvom prolasku, podaci klase 1 imat će vrijednost 1, a klase 2 i 3 vrijednost 0, pa dobijemo granicu odluke između klase 1 i ostalih dviju

- u drugom prolasku, podaci klase 2 imat će vrijednost 1, a klase 1 i 3 vrijednost 0, pa dobijemo granicu odluke između klase 2 i ostalih dviju
- u trećem prolasku podaci klase 3 imat će vrijednost 1, a klase 1 i 2 vrijednost 0, pa dobijemo granicu odluke između klase 3 i ostalih dviju.

Rezultat su tri funkcije hipoteze koje daju vrijednost sigmoidne funkcije između 0 i 1. U ovom slučaju nema smisla određivati binarni izlaz kao kod klasifikacije u dvije klase koristeći granicu na vrijednosti 0,5. Izlazna vrijednost sigmoidne funkcije *h(x)* u ovoj situaciji može imati smisao vjerojatnosti da podatak pripada klasi za koju je hipoteza istrenirana.



Slika 6-14. Ilustracija algoritma „jedan protiv svih"

Za slučaj *m* klasa to možemo iskazati izrazom:

$$h_{\boldsymbol{\theta}}^{(i)}(\boldsymbol{x}) = P(\, y = i|\boldsymbol{x}\, ;\, \boldsymbol{\theta}) \; za \; i = 1,2, \dots, m \qquad (6\text{-}16)$$

Odluka se donosi na temelju ove vjerojatnosti, odabirom one klase čija odgovarajuća hipoteza ima najveću vrijednost. To znači da kod logističke regresije u više klasa na izlazu nemamo binarni kvantifikator, već komparator.

Pogledajmo primjer za situaciju sa slike 6-14. Pretpostavimo da smo za jedan konkretni uzorak $\boldsymbol{x}$ =(x_1, x_2) dobili u tri prolaska sljedeće:

$h_{\boldsymbol{\theta}}^{(1)}(\boldsymbol{x}) = P(\, y = 1|\boldsymbol{x}\, ;\, \boldsymbol{\theta}) = 0{,}6$

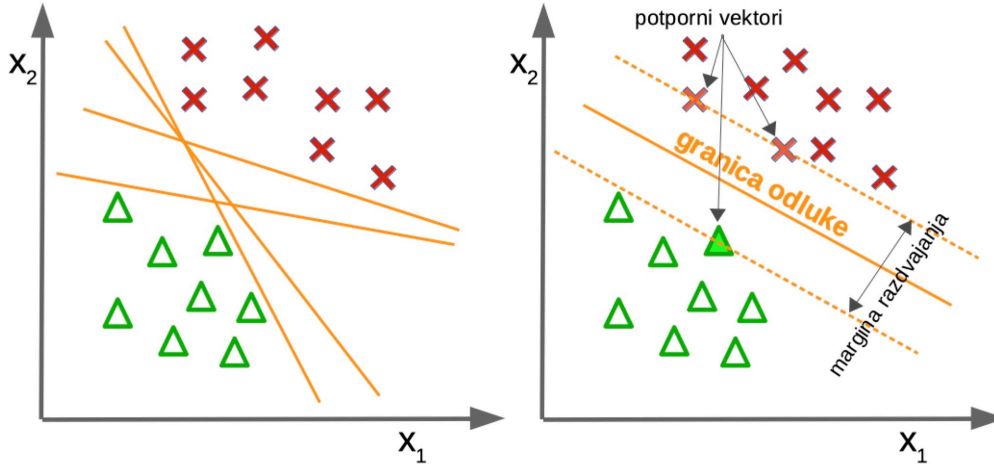
$h_{\boldsymbol{\theta}}^{(2)}(\boldsymbol{x}) = P(\, y = 2|\boldsymbol{x}\, ;\, \boldsymbol{\theta}) = 0{,}4$

$h_{\boldsymbol{\theta}}^{(3)}(\boldsymbol{x}) = P(\, y = 3|\boldsymbol{x}\, ;\, \boldsymbol{\theta}) = 0{,}3$

Zaključujemo da ćemo taj uzorak pridijeliti klasi 1 zato što u prvom prolasku hipoteza ima najveću vrijednost 0,6.

6.3.7 Stroj s potpornim vektorima

Stroj s potpornim vektorima (engl. *SVM – Support Vector Machine*) je sličan logističkoj regresiji i također se koristi kod klasifikacije u linearno odvojive klase binarnom klasifikacijom. Ideja je pronaći takvu ***granicu odluke*** koja ima najveću udaljenost od najbližih uzoraka koji pripadaju pojedinim klasama. Razliku između standardne granice odluka i stroja s potpornim vektorima u slučaju dviju značajki x_1 i x_2 prikazuje slika 6-15.



Slika 6-15. Razlika između standardnih linearnih granica odluke (lijevo) i granica odluke stroja s potpornim vektorima (desno). Kod logističke regresije traži se pravac koji maksimizira vjerojatnost da se slučajno odabrani uzorak pravilno klasificira, a kod stroja s potpornim vektorima pravac koji ima najveću marginu razdvajanja

Kod stroja s potpornim vektorima traži se takva granica odluke koja ima najveću ***marginu razdvajanja***. Uzorci koji se nalaze najbliže margini razdvajanja zovu se ***potporni vektori***.

Kod logističke regresije nastoji se maksimizirati vjerojatnost da se slučajno odabrani uzorak pravilno klasificira. Što je uzorak dalje od granice odluke, to je funkcija cijene bolja, a na granicu odluke utječu svi uzorci. Kod stroja s potpornim vektorima nastoji se maksimizirati margina razdvajanja, pa su važni jedino uzorci najbliži margini razdvajanja (potporni vektori). Svaki od ova dva postupka klasifikacije ima svoje prednosti i nedostatke. Stroj s potpornim vektorima je nešto manje osjetljiv na uzorke koji upadaju u ***suprotne klase*** (tzv. „*outliers*"), a iskustva kažu da daje i bolje rezultate ako imamo malo značajki *(n)*, a velik broj uzoraka za treniranje *(m)*.

Ako se uzorci mogu idealno klasificirati (klase su linearno odvojive), moguće je povući pravce paralelne s granicom odluke na kojima će ležati potporni vektori. U tom linearnom slučaju funkcija cijene stroja s potpornim vektorima definira se izrazom:

$$Cost(h_{\boldsymbol{\theta}}(x), y) = \begin{cases} max\left(0, 1 - h_\theta(x)\right), za\ y = 1 \\ max\left(0, 1 + h_\theta(x)\right), za\ y = 0 \end{cases} \tag{6-17}$$

Kada sumiramo po svim podacima, dobije se funkcija cijene hipoteze za određenu kombinaciju parametara $\boldsymbol{\theta}$:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} \cdot Cost_1\left(h_{\boldsymbol{\theta}}\left(x^{(i)}\right), y^{(i)}\right) + \left(1 - y^{(i)}\right) \cdot\ Cost_1\left(h_{\boldsymbol{\theta}}\left(x^{(i)}\right), y^{(i)}\right)\right) =$$
$$= -\frac{1}{m}\left[\sum_{i=1}^{m}\left(y^{(i)} \cdot max\left(0, 1 - h_{\boldsymbol{\theta}}\left(x^{(i)}\right)\right) + (1 - y^{(i)}) \cdot max\left(0, 1 + h_{\boldsymbol{\theta}}\left(x^{(i)}\right)\right)\right)\right] \tag{6-18}$$

Lako je uočiti da postoji velika sličnost s funkcijom cijene logističke regresije opisane jednadžbama (6-13) i (6-14), s tim da kod stroja s potpornim vektorima maksimiziramo marginu razdvajanja. Kao i kod logističke regresije na ovu funkciju cijene možemo primijeniti i regularizaciju (poglavlje 6.3.5).

Ostaje nam još za pronaći takvu kombinaciju parametara $\boldsymbol{\theta}$ koji minimiziraju ovu funkciju cijene $min_{\theta}\ J(\boldsymbol{\theta})$. Jedan od postupaka je svakako i spust gradijenta s tim da se obično koristi ***stohastički spust gradijenta*** (engl. *Stohastic Gradient Descent*) ili ***stohastički spust sub-gradijenta*** (engl. *Stohastic Sub-Gradient Descent*). Za više detalja čitatelje upućujemo na dodatnu literaturu.

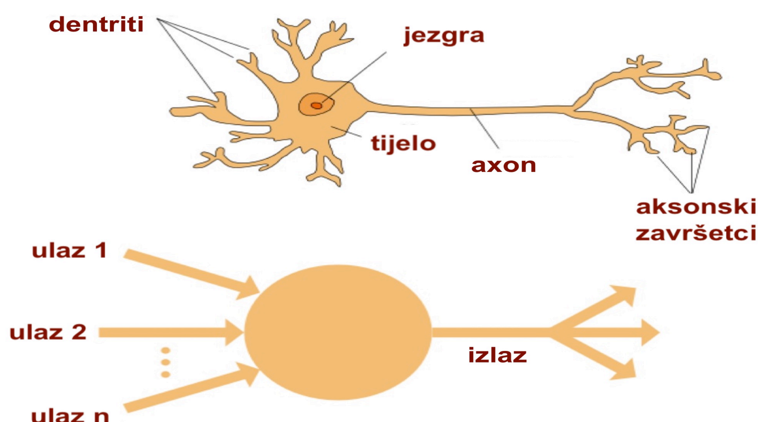
Osim linearnog stroja s potpornim vektorima postoji i nelinearni koji se koristi u slučajevima kada se ne postoji linearna odvojivost klasa. Od linearnog se razlikuje po tome što se uvodi tzv. ***funkcija kernela*** (engl. *Kernel Function*).

6.3.8 Umjetne neuronske mreže i duboko učenje

Duboko učenje (engl. *Deep Learning*) u posljednje vrijeme budi interes javnosti i pokazuje izvrsne rezultate u područjima poput računalnog vida, obrade prirodnog jezika, prepoznavanja govora, igranja igara i sl. Duboko učenje oslanja se na ideju ***umjetnih neuronskih mreža*** (engl. *ANN – Artificial Neural Networks*) koje oponašaju rad biološkog neuronskog (živčanog) sustava. Umjetne neuronske mreže uveli su 1943. godine neuroznanstvenik ***Warren S. McCulloch*** i logičar ***Walter Pitts***[95], a popularnost im raste nakon što je 1949. godine ***Donald Hebb*** postavio temeljni zakon učenja neurona poznat kao ***Hebbovo pravilo*** koje se kratko opisuje rečenicom: *„Neuroni koji se zajedno aktiviraju se i povezuju"*. Prava primjena umjetnih neuronskih mrež omogućena je tek razvojem računala, dostupnošću velikih količina podataka i novih topologija mreža koje su omogućile stvaranje kompleksnijih mrežnih struktura.

Umjetni neuron (perceptron)

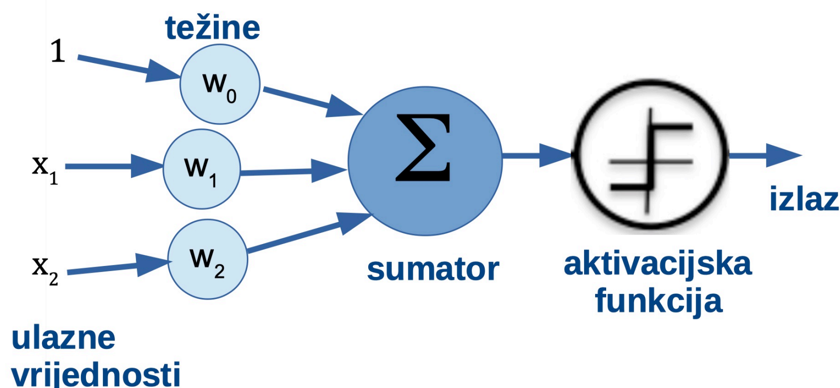
Istraživanjem neuronskog sustava bioloških organizama bave se neuroznanstvenici koji su otkrili da se neuronski sustav sastoji od međusobno povezanih neurona (živčanih stanica). Neuroni (slika 6-16) se sastoje od ***tijela*** (*soma*), ***ulaznih niti*** (*dendrita*) i ***izlaznih niti*** (*aksona*). Dendriti primaju električne impulse iz okoliša ili drugih stanica, tijelo ih promijeni u drugi oblik (obradi), te se onda aksonima provode sljedećoj razini neurona. Na taj način živčani sustav bioloških organizama prenosi informacije koje prikupljaju biološka osjetila i upravlja radom cjelokupnog organizma. Umjetni neuroni su matematičke funkcije koje oponašaju rad bioloških neurona. Dendrite predstavljaju ulazni podaci, a aksone izlazni podaci. Tijelo neurona provodi funkciju sumiranja ulaznih vrijednosti.



Slika 6-16. Biološki neuroni su inspiracija za računalnu tvorevinu – umjetni neuron

[95] Inicijalni rad McCullocka i Pittsa objavljen je u članku *„A logical calculus of the ideas immanent in nervous activity"* u ***Bulletin of Mathematical Biophysics*** 5:115-133, 1943. g.

Perceptron je računalna implementacija umjetnog neurona. On je samostalna računalna jedinica koja ulazne vrijednosti množi težinama, zbraja ih i računa ukupnu vrijednost sume na koju zatim djeluje aktivacijskom funkcijom koja uspoređuje vrijednost sume s graničnom vrijednosti koju nazivamo ***prag*** (engl. *Threshold*), te na temelju te usporedbe određuje vrijednost izlaznog signala. Slika 6-17 prikazuje grafičku interpretaciju perceptrona.



Slika 6-17. Računalna interpretacija perceptrona (umjetnog neurona)

Usporedimo li ga s grafičkim prikazom algoritama linearne i logističke regresije (Slike 6-4 i 6-13), lako je uočljiva sličnost. Perceptron je u biti algoritam linearne regresije kojem je na izlazu dodana ***aktivacijska funkcija*** (engl. *Activation Function*), odnosno logistička regresija koja nema blok sa sigmoidnom funkcijom. Aktivacijska funkcija je funkcija jediničnog skoka koja za ulaznu vrijednost veću od 0 daje 1, a za ulaznu vrijednost manju od 0 daje -1. Ovdje je važno uočiti da su izlazi 1 i -1 iako se radi o binarnoj klasifikaciji kod koje se kao izlazne vrijednosti uobičajeno koriste 0 i 1. Razlog je algoritam učenja perceptrona koji opisujemo u nastavku.

Ulazna vrijednost računa se na isti način kao i hipoteza linearne regresije, s tim da je kod neuronskih mreža uobičajeno koristiti druge oznake:

$$h_w(x) = w_0 + w_1x_1 + w_2x_2 \qquad (6\text{-}19)$$

Težina w_0 ima poseban značaj. Ona određuje prag iznad kojeg suma $(w_1x_1 + w_2x_2)$ na izlazu daje 1, pa se zato i naziva ***faktor korekcije*** (engl. *Bias*) i često označava s b. Na primjer, za $w_0 = -2$ perceptron će na izlazu imati 1 ako je $(w_1x_1 + w_2x_2) > 2$.

Kao i u prethodnim slučajevima projektirati perceptron znači odrediti težine w_0, w_1 i w_2. Postupak koji se koristi je vrlo sličan spustu gradijenta koji se koristio kod linearne i logističke regresije, ali ne potpuno isti. Početnu ideju je 1957. godine predložio američki psiholog ***Frank Rosenblatt*** (1928. – 1971.) i nazvao ju ***perceptronsko pravilo*** (engl. *Percepton Rule*). Sastoji se od toga da se krene od nekih malih, proizvoljno odabranih vrijednosti težina. Za ***svaki ulazni podatak*** (x_1,x_2) težina se korigira već poznatom jednadžbom iz spusta gradijenta:

$$w_j := w_j - \Delta\, w_j^{(i)} = w_j - \alpha \cdot (h_w^{(i)}(\boldsymbol{x}) - y^{(i)}) \cdot x_j^{(i)} \qquad (6\text{-}20)$$

Osnovna razlika između spusta gradijenta i perceptronskog pravila je u riječima „*za svaki ulazni podatak*”. Kod spusta gradijenta težine se korigiraju nakon cijele sekvence podataka za treniranje. Kod proračuna novih vrijednosti težina, razlika između dobivenog izlaza i željenog izlaza $(h_w^{(i)}(\boldsymbol{x}) - y^{(i)}) \cdot x_j^{(i)}$ se najprije sumira za sve ulazne podatke (i = 1, ... , m), pa tek onda množi s parametrom brzine učenje α i oduzima od postojeće vrijednosti težina. Kod perceptrona se korekcija težina radi za svaki ulazni podatak. Zbog toga se ponekad algoritam perceptronskog pravila naziva ***inkrementalni spust gradijenta*** (engl. *Incremental Gradient Descent*).

Pogledajmo primjer. Kod perceptrona hipoteza i željeni izlaz mogu imati samo vrijednosti 1 ili -1. U slučaju da perceptron pogodi izlaz nema korekcije težina:

$$w_j := w_j - \alpha \cdot (1-1) \cdot x_j^{(i)} = w_j - 0$$

$$w_j := w_j - \alpha \cdot (-1-(-1)) \cdot x_j^{(i)} = w_j - 0$$

U slučaju da perceptron ne pogodi izlaz, za negativne vrijednosti stvarnog izlaza korekcija će biti negativna, a za pozitivne pozitivna:

$$w_j := w_j - \alpha \cdot (1-(-1)) \cdot x_j^{(i)} = w_j - \alpha \cdot 2 \cdot x_j^{(i)}$$

$$w_j := w_j - \alpha \cdot (-1-1) \cdot x_j^{(i)} = w_j + \alpha \cdot 2 \cdot x_j^{(i)}$$

Veličina promjene ovisi o parametru brzine učenja i ulaznoj vrijednosti. Postupak je vrlo jednostavan i može se koristiti za jednostavnu binarnu klasifikaciju metodom nadziranog učenja. Ograničenje mu je konvergencija. Namještanje težina konvergira samo ako su klase linearno odvojive. Ako se klase ne mogu razdvojiti linearnom granicom odluke, promjena parametara neće konvergirati. Parametri će oscilirati pa je nužno ograničiti maksimalan broj prolazaka kroz podatkovni skup za učenje i/ili postaviti prag za broj toleriranih pogrešnih klasifikacija. Problem linearne odvojivosti može se riješiti i unapređenjem arhitekture perceptrona na način da se uzme drugačija aktivacijska krivulja koja na izlazu ne daje binarne podatke -1 ili 1, već vrijednost iz intervala [-1, 1]. Tipičan primjer je funkcija slična sigmoidnoj funkciji iz logističke regresije, samo što je spuštena tako da daje izlazne vrijednosti u intervalu [-1, 1], a kreira se na primjer pomoću ***hiperbolnog tangesa***:

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (6\text{-}21)$$

Umjetni neuron s ovakvom kontinuiranom aktivacijskom funkcijom uobičajeno se zove ***sigmoidni neuron*** (engl. *Sigmoid Neuron*).

Osim aktivacijske funkcije hiperbolnog tangensa, u umjetnim neuronskim mrežama koriste se i druge aktivacijske funkcije, pa i originalna sigmoidna funkcija (koja se ponekad naziva i ***logistička aktivacijska funkcija***). Ona daje izlaznu vrijednost u intervalu [0 , 1], što ponekad stvara problem u procesu učenja. U dubokom učenju posebno je popularna pojednostavljena verzija logističke funkcije koja se naziva ***ispravljena linearna funkcija*** (engl. *ReLU – Rectified Linear Unit*):

$$f(x) = \begin{cases} 0, za\ x \leq 0 \\ x, za\ x > 0 \end{cases} \quad (6\text{-}22)$$

koja daje izlaznu vrijednost u intervalu [0 , ¥]. Kako za sve negativne vrijednosti na izlazu daje 0, i ona ponekad ima probleme u procesu učenja, pa se primjenjuje i njena modificirana verzija ***parametarska ReLU*** (engl. *Parametric ReLU)*:

$$f(x) = \begin{cases} \alpha x, za\ x \leq 0 \\ x,\ za\ x > 0 \end{cases} \quad (6\text{-}23)$$

Ako je $\alpha = 0,01$ naziva se ***propustljiva ReLU*** (engl. *Leaky ReLU).*

Spajanjem više perceptrona na način da izlazni signali neurona početnog sloja budu ulazni signali neurona sljedećeg sloja dobije se ***višeslojni perceptron*** (engl. *Multi Layered Perceptron*) koji sada već spada u umjetne neuronske mreže.

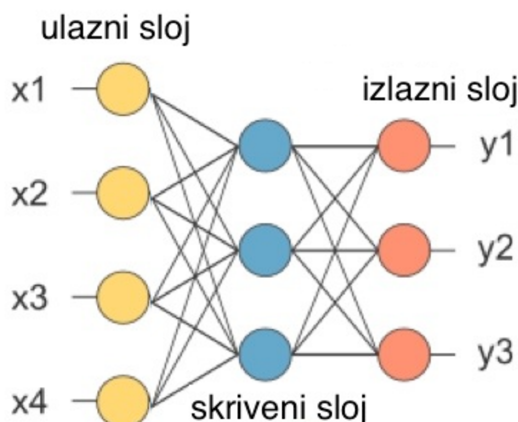
Umjetna neuronska mreža

Umjetne neuronske mreže (engl. *ANN – Artificial Neural Network*) su mreže umjetnih neurona. Umjetna neuronska mreža spada u nadzirane ili polunadzirane metode strojnog učenja. Mreža se sastoji od umjetnih neurona, najčešće poredanih u slojeve, tako da razlikujemo:

- ***ulazni sloj neurona*** koji preuzimaju ulazne signale

- ***skrivene slojeve*** koji prenose signale i
- ***izlazni sloj*** koji daje izlazne zaključke.

Struktura prikazana na slici 6-18 obično se naziva ***unaprijedna neuronska mreža*** (engl. *Feedforward Neural Network*).



Slika 6-18. Unaprijedna umjetna neuronska mreža s jednim skrivenim slojem

Svaki neuron ima svoje težinske faktore s kojima se množe njegovi ulazni signali, sumator i aktivacijsku funkciju kojom se računa njegov izlazni signal. Projektirati umjetnu neuronsku mrežu znači:

- ***odabrati njenu arhitekturu*** (broj umjetnih neurona, oblik aktivacijske krivulje, broj slojeva, način međusobnog povezivanja umjetnih neurona) i
- ***trenirati (naučiti)*** umjetnu neuronsku mrežu na način da se odrede vrijednosti svih težina tako da mreža za ulaze iz skupa podataka za treniranje daje izlaze što sličnije rezultatima iz skupa za treniranje.

Kod treniranja umjetne neuronske mreže najčešće se koristi postupak ***propagacije pogreške unatrag*** (engl. *Backpropagation*). To je iterativni postupak kojim se optimiziraju vrijednosti težinskih faktora neurona kako bi na izlazu dobili vrijednosti iz skupa za treniranje. Pri tome se najčešće koriste različite inačice postupka spusta gradijenta koji smo detaljno upoznali.

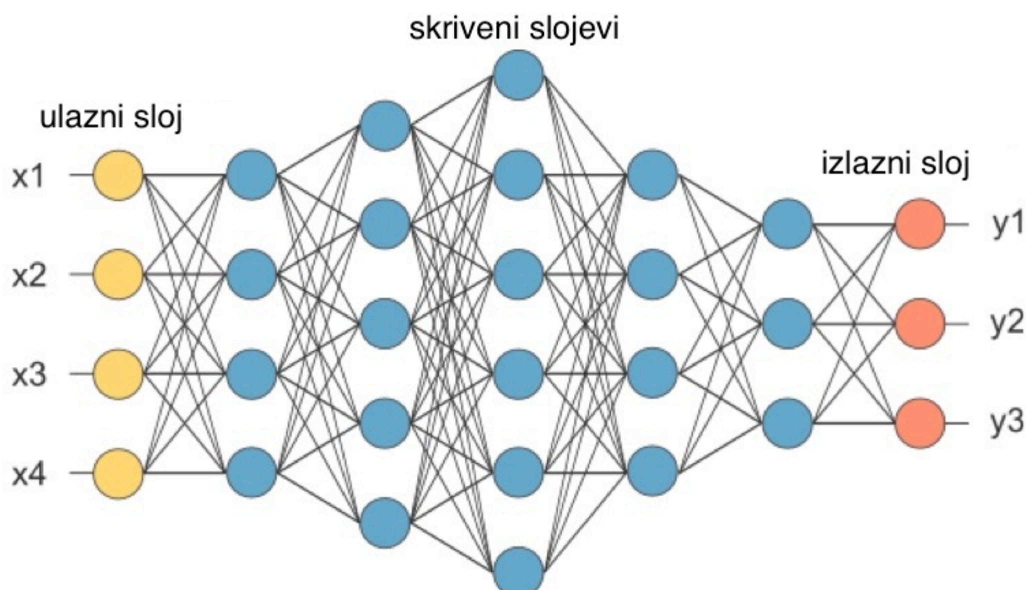
Neuronske mreže u početku su se sastojale samo od jednog skrivenog sloja. Dugotrajan i složen postupak treniranja mreže ograničavao je praktičnu primjenu većeg broja slojeva i kompleksnijih struktura. Današnje računalne arhitekture, najviše zahvaljujući paralelnoj obradi podataka, omogućile su realizaciju neuronske mreže s više skrivenih slojeva koje se nazivaju duboke neuronske mreže.

Duboke neuronske mreže i duboko učenje

Duboke neuronske mreže (engl. *Deep Neural Networks*) sastoje se od više skrivenih slojeva, ali i koriste naprednije strukture od višeslojnog perceptrona. Jednu duboku unaprijednu neuronsku mrežu s pet skrivenih slojeva prikazuje slika 6-19.

Osim unaprijedne arhitekture, danas se koriste i ***rekurentne neuronske mreže*** *(engl. Reccurent Neural Networks*) kod kojih protok informacija među neuronima ide u svim smjerovima. Ovakve mreže su pokazale dobre rezultate u obradi prirodnog jezika. Tu su i ***konvolucijske neuronske mreže*** (engl. *Convolutional Neural Networks)* koje koriste trodimenzionalnu strukturu pogodnu za obradu i analizu digitalnih slike i daju odlične rezultate na području kognitivnog računalnog vida, posebno u prepoznavanja objekata na slici i odnosa među njima (prepoznavanje scene). Ilustraciju duboke neuronske mreže za prepoznavanje lica već smo prikazali na slici 1-11.

Danas duboke neuronske mreže nalaze brojne praktične primjene, a projektiraju se obično korištenjem posebnih ***programskih okvira*** (engl. *Frameworka*) kao što su ***TensorFlow***, ***Keras*** i ***PyTorch***.



Slika 6-19. Duboka unaprijedna neuronska mreža s pet skrivenih slojeva

Projektirati duboku neuronsku mrežu znači definirati ***hiperparametre*** koji predstavljaju vanjske parametre mreže. Hiperparametri se dijele u dvije grupe:

- ***hiperparametri vezani za strukturu mreže***:
  - broj skrivenih slojeva – duboko učenje obično koristi 3 – 10 skrivenih slojeva
  - odumiranje – koliki će broj neurona slučajnim odabirom odumrijeti kako bi se spriječilo pretreniranje
  - aktivacijska funkcija – koju će aktivacijsku funkciju neuroni koristiti (linearnu, sigmoidnu, tanh, softmax, ReLU itd.)
  - inicijalne težine za prvu iteraciju treniranja – tipično se postavljaju na 0 ili slučajnim odabirom, ali nekada se može koristiti heuristika – npr. kod tanh aktivacijske funkcije može se koristi tzv. Xavierova inicijalizacija[96].
- ***hiperparametri vezani za algoritam učenja***:
  - brzina učenja – korak učenja kod spusta gradijenta
  - broj epoha učenja, iteracija i veličina serije jedne iteracije ***„backpropagation"*** učenja
  - algoritam optimizacije – temeljni algoritam koji se koristi je stohastički spust gradijenta, no tipično se još koriste[97] i ***miniserija spusta gradijenta*** (engl. *Mini-batch Gradient Descent*), ***momentum algoritam*** (koji čeka da se težina osvježi te je ažurira drugi put koristeći delta vrijednost što se pokazalo da ubrzava učenje), ***RMS Prop*** (engl. *Root-Mean-Square Propagation*), ***AdaM*** (engl. *Adaptive Momentum*), ***AdaGrad***, ***AdaDelta***, ***Nesterov ubrzani gradijent***.

Programski okviri (engl. *Frameworka*), kao što su prije spominjani ***TensorFlow***, ***Keras*** i ***PyTorch***, omogućavaju relativno jednostavno podešavanje ovih hiperparametara, te pokretanje algoritma treniranja na skupu podataka za treniranje. Programerski okvir inicijalizira mrežu željene

[96] Za detalje pogledati na primjer https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/

[97] Za detalje pogledati na primjer https://d2l.ai/chapter_optimization/index.html

strukture te pokreće algoritam treniranja. Po završetku treniranja, dobijemo model s istreniranim parametrima za svrhu određenu podacima za treniranje.

Praktične primjene također često koriste i ***prijenosno učenje*** (engl. *Transfer Learning*). Prijenosno učenje odnosi se na korištenje već stečenog znanja i njegovog nadograđivanja. Naime, nekada se za inicijalne težine koriste podaci već istrenirane duboke neuronske mreže te se optimiziraju manjim skupom podataka za treniranje. Druga mogućnost je koristiti već istreniranu neuronsku mrežu za generičke namjene, a trenirati samo dodatne slojeve za specifičnu namjenu.

Korištenje biblioteka i programerskih okvira olakšalo je implementaciju projekata za industrijsku primjenu dubokog učenja. To je rezultiralo time da danas mnogi praktičari duboko učenje svode na model ***„crne kutije"*** (engl. *Black Box*) bez razumijevanja načina rada internih procesa. To i nije baš pravi put, zato što se zadovoljavajući rezultati mogu dobiti samo uz dobro razumijevanje domenskog znanja, mehanizma treniranja i zaključivanja dubokih neuronskih mreža.

6.4 Nenadzirano učenje

Nadzirano učenje ima za cilj pronaći hipotezu koja se na najbolji način uklapa u ulazne označene podatke, dok nenadzirano učenje ima za cilj pronalaženja strukture u neoznačenim podacima. Nenadziranog učenja je posebno značajno u današnje vrijeme, zato što su neoznačeni podaci lako dostupni i jeftini (otvoreni web, podaci sa senzora, IoT). Skupo ih je označavati, pa su postupci koji ne zahtijevaju označene podatke vrlo važni. Nenadzirano učenje se može podijeliti po različitim kriterijima, a mi ćemo ovdje navesti podjelu po ciljevima, pa razlikujemo:

- ***klasteriranje*** (postupak grupiranja podataka po sličnim značajkama)
- ***smanjenje dimenzionalnosti*** (postupak kojim se matematičkim postupcima identificiraju najbitnije značajke podataka te se smanjuje njihova dimenzionalnost) i
- ***detekcija anomalija*** (koristi se za prepoznavanje podataka koji se ne uklapaju u očekivane podatke).

6.4.1 Klasteriranje

Klasteriranja (engl. *Clustering*) ima za cilj grupirati podatke u grupe ili klastere prema sličnosti određenih značajki. Slični podaci svrstavaju se u iste grupe. Ovakav postupak koristan je u mnogim segmentima industrijske primjene strojnog učenja, a neki od njih su:

- ***segmentacija tržišta*** (grupe kupaca koje imaju slične afinitete)
- ***analiza društvenih mreža*** (Facebook, Twitter itd. imaju velike količine pohranjenih podataka o korisnicima, pa se klasteriranje koristi za prepoznavanje sličnih korisnika)
- ***klasteri servera*** (alokacija resursa za distribuirano računanje prema sličnim karakteristikama i blizini)
- ***astronomski podaci*** (razumijevanje formiranja galaksija) itd.

Najpoznatiji algoritam za klasteriranje je algoritam k-sredina.

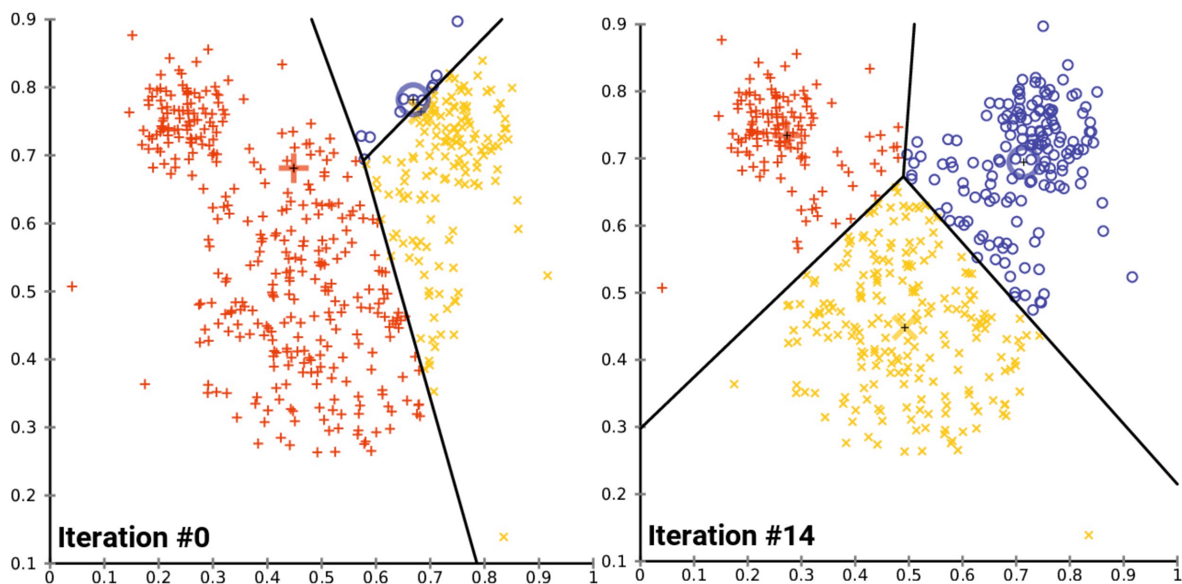
Algoritam k-sredina

Algoritam k-sredina (engl. *k-means Algorithm*) je algoritam za automatsko grupiranje podataka u koherentne grupe. Najpoznatiji algoritam za klasteriranje, a popularnost može zahvaliti tome što je intuitivan i jednostavan za implementaciju. Da bismo ga primijenili, potrebno je poznavati značajke podataka koje želimo razdvojiti u grupe i broj grupa (K) na koji ćemo podatke razdvojiti. Nakon provođenja postupka dobijemo informacije o formiranju K klastera, a svaki od njih je opisan sa svojim prosječnim vrijednostima značajki. Postupak je sljedeći:

- Na početku ili svjesnim vlastitim odabirom ili slučajnim odabirom postavimo K početnih centara klastera μ_k, $k=1,\ldots,K$.

- Za svaki podatak x_i provjerimo njegovu udaljenost od centara svih K klastera te primjenom operacije *argmin f(.)* koja vraća element skupa za koji je funkcija *f(.)* minimalna taj podatak x_i pridružimo klasteru kojem je najbliži. Pri tome se obično koristi Euklidska udaljenost.
- Prolaskom kroz sve podatke, dobit ćemo prvu verziju razdiobe podataka na klastere.
- Sada je potrebno izračunati nove centre klastera, na način da se izračuna srednja vrijednost svih podataka koji su tom klasteru pridruženi.
- Nakon što su se centri pomakli, postupak se ponavlja. Ponovno se prolazi kroz sve podatke te se pojedinačni podaci pridjeljuju najbližem centru te se izračunavaju novi centri klastera sve dok ne dobijemo stacionarno stanje što znači da se centri klastera više ne mijenjaju.

Primjer klasteriranja postupkom k-sredina u tri klase na temelju dviju značajki prikazuje slika 6-20. Prikazana je početna situacija i podjela u klase nakon 14 iteracija.



Slika 6-20. K-means klasteriranje u tri klase na temelju dviju značajki na početku i nakon 14 iteracija[98]

Pseudokod algoritma k-sredina daje algoritam 6-2.

Algoritam 6-2. Pseudokod algoritma za klasteriranje u K klastera

```
//Definirati broj klastera K
K
//Inicijalizirati slučajnim odabirom početne centre K klastera (k = 1 , K)
μ1, ... , μK
Repeat
//Za svih m ulaznih podataka izračunaj pripadanje γij jednom od klastera na temelju Euklidske udaljenosti
for (i = 1 to m) {
γik = {1 if k=argmink ||xi- μk||2 else 0} }
```

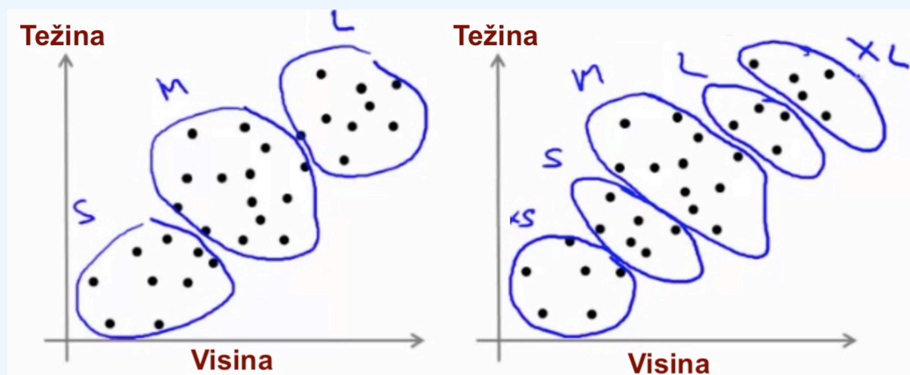
[98] Slika je s https://commons.wikimedia.org/wiki/File:K-means_convergence.gif uz GNU Free Documentation License.

```
//Izračun novih centara klastera
for (k = 1 to K) {
nk = SUM(i=1 to m)(γik)   //# podataka klastera k
    μk =    (1/nk)SUM(i=1 to n)( γikxi)  }    //novi centri klastera
  until Covergence
```

Nedostaci algoritma k-sredina

Pet je temeljnih nedostatka algoritma k-sredina:

1. Iako je intuitivan, i daje dobre rezultate za većinu praktičnih primjena, kritičari algoritma k-sredina zamjeraju mu *nedostatak matematičke osnove*. Naime, algoritam k-sredina nije nastao temeljem matematičkog izvoda, već intuitivnog postupka.
2. *Rezultat algoritma – klasteri*, ovise o odabranim početnim točkama centara klastera. Ako ponovimo postupak s drugačije odabranim početnim centrima klastera, rezultati se mogu razlikovati. To znači da je moguće da algoritam pronađe lokalni optimum, a ne globalni optimum – najbolji izbor centara klastera. Šanse za pronalazak lokanog optimuma mogu se umanjiti tako da se cijeli postupak algoritma ponovi više puta, ali s različitim izborom početnih točaka centara klastera. Većim brojem ponavljanja uočit će se postojanje globalnog optimuma kod više rezultata, dok će se lokalni optimum pojaviti samo kod pojedinih izbora početnih točaka. Postavlja se pitanje kako ocijeniti koja je najbolja razdioba klastera. Intuitivno rješenje je da je bolja razdioba ona u kojoj su podaci najbolje grupirani oko svog centra. To možemo kvantitativno izraziti mjerom srednje kvadratne udaljenosti podatka od centra njegovog klastera. Dakle, nakon iterativnog provođenja k-sredina klasteriranja s različito odabranim početnim centrima odaberemo rezultat s najmanjom vrijednošću ove mjere.



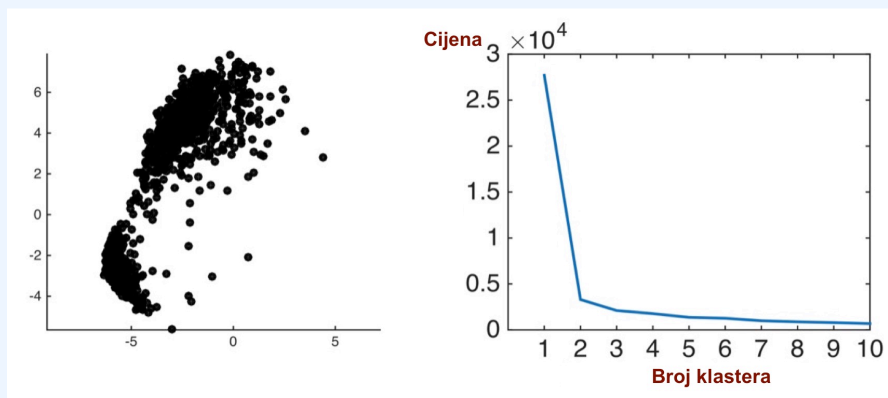
Slika 6-21. Klasteriranje podataka u klastere ovise o odabiru početnih centroida. Na objema slikama isti su podaci odnosa visine i težine koji su na lijevoj slici klasterirani u tri klastera veličine majica (S, M i L), a na desnoj slici u pet klastera veličina (XS, S, M, L, XL)[99]

3. *Nerazdvojivi klasteri* – u slučaju da su podaci takvi da nema jasne granice razdvajanja među grupama (podaci su kontinuirani), rezultati algoritma k-sredina ovisit će o odabiru početnih točaka. U ovom slučaju ni intuitivno ne možemo razdvojiti podatke na klastere ovakvim postupkom.

4. *Poznavanje broja klastera* – algoritmu k-sredina kao ulazni podatak potreban je broj očekivanih klastera. Nekada imamo samo podatke, ali ne znamo koliko različitih skupina

[99] Slika je s OpenCV Python Tutoriala

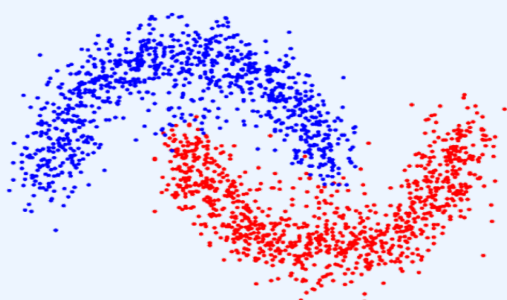
očekujemo u tim podacima. Preporučeni način pronalaska broja klastera algoritmom k-sredina je tzv. ***metoda lakta*** (engl. *Elbow Method*).



Slika 6-22. Metoda lakta (Elbow Method) kod odabira optimalnog broja klastera

Ova je metoda učinkovita samo ponekad, ali uvijek vrijedi pokušati. Postupak se sastoji od uzastopnog ponavljanja algoritma k-sredina za različite brojeve klastera. Za iste podatke ponovimo postupak algoritma k-sredina za pronalazak jednog centra, zatim 2 centra, pa 3 centra itd. Za svaki od rezultata izračunamo ukupnu sumu kvadratne udaljenosti podataka od najbližeg centra (ova vrijednost predstavlja cijenu rezultata) te prikažemo grafički ovisnost cijene o broju klastera. Ako se može primijeniti metoda lakta, graf bi trebao izgledati kao na slici 6-22. Kada dođemo do broja klastera u koje se podaci zapravo i mogu podijeliti, vrijednosti funkcije cijene prijeći će iz naglog pada povećanjem broja klastera do manjeg pada. Drugim riječima, povećanjem broja klastera nećemo dobivati značajan pad funkcije cijene. Na grafičkom prikazu ova će se vrijednost odraziti kao *„lom"* ili *„lakat"* (engl. *Elbow*). Broj klastera kojem odgovara taj lom je najbolji kandidat za broj klastera skupa podataka s kojima radimo. Na Slici 6-21 najbolji broj klastera je 2.

5. *Klasteri bez centra.* Ponekad imamo podatke koje želimo razdvojiti, ali ne temeljem udaljenosti od centra, već nečega drugog. Algoritam k-sredina tada nam nije od pomoći. Primjer pokazuje slika 6-23 na kojoj imamo prikazane podatke koji se ne mogu podijeliti na temelju udaljenosti od centra. U ovakvim se situacijama koriste drugi algoritmi klasteriranja, na primjer ***spektralno klasteriranje*** (engl. *Spectral Clustering*).



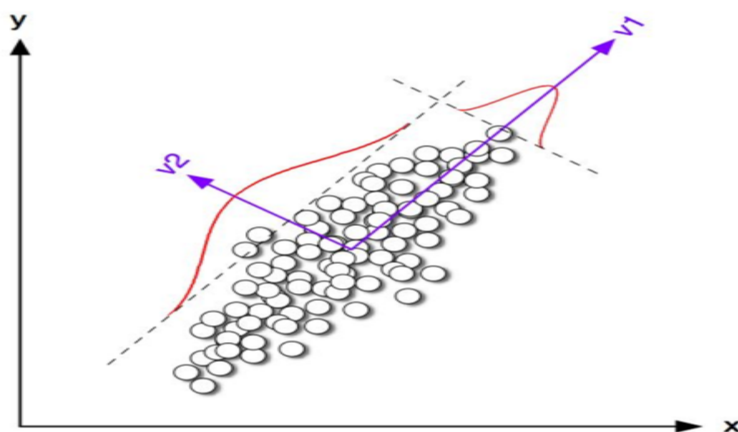
Slika 6-23. Spektralno klasteriranje

6.4.2 Smanjenje dimenzionalnosti

Dimenzija skupa podataka određena je brojem značajki pomoću kojih su podaci opisani. Ponekad nam nije problem prikupiti velik broj značajki, ali skup s većim brojem značajki teže je obrađivati.

Algoritmi za smanjenje dimenzionalnosti specijalizirani su za pronalazak značajki koje možemo ukloniti iz skupa za treniranje, a da pritom izgubimo najmanji dio informacije. Razlozi za smanjivanje dimenzionalnosti su razni, no često se radi o potrebi da podatke vizualiziramo. Ljudi mogu vizualizirati podatke u najviše 3 dimenzije, pa svođenjem podataka na 2 ili 3 značajke moguće ih je vizualizirati. Drugi razlog za reduciranje dimenzija podataka je sa ciljem ubrzavanja postupka treniranja nekim drugim algoritmima. Npr. podatke od 1000 značajki reduciramo na 100 značajki, te takve podatke koristimo za treniranje linearne regresije. Bez obzira koji je razlog smanjenja dimenzionalnosti, cilj ovog postupka je pronaći koje dimenzije (značajke) možemo ukloniti kako bi se zadržalo što više informacija.

Reduciranje broja značajki u velikom broju slučajeva ne znači i odbacivanje pojedinih značajki. Nekada su podaci takvi da se neke značajke mogu lako prikazati kao linearna kombinacija drugih značajki. Takvi podaci mogu se odbaciti jer su redundantniji. Npr. na slici 6-24 prikazan je skup podataka određen s dvije značajke.



Slika 6-24. Dvodimenzionalni (2-D) podaci koji se mogu sažeti u jednu dimenziju

Promotrimo li ove podatke prikazane u koordinatnom *(x, y)* sustavu, lako možemo uočiti alternativni koordinatni sustav *(v1, v2)* u koji ove podatke može jednostavno transformirati. Nadalje možemo primijetiti da su u tom koordinatnom sustavu podaci prilično široko razasuti po dimenziji *v1*, dok se po dimenziji *v2* nalaze u veoma uskom pojasu. Drugim riječima svi podaci imaju približno jednaku vrijednost dimenzije *v2* te nam vrijednost dimenzije *v2* ne nosi baš neku informaciju i možemo je izostaviti. Time smo podatke iz 2D prostora *(x, y)* prebacili u 1D prostor (*v1*). Najpoznatiji algoritam za smanjivanje dimenzionalnosti je algoritam analize glavnih komponenti.

Analiza glavnih komponenti (PCA)

Algoritmom ***analiza glavnih komponenti*** (engl. *PCA – Principal Component Analysis*) prikazujemo izvorne podatke pomoću manjeg skupa linearnih kombinacija značajki. Ovaj se algoritam temelji na analizi matrica i koristi brojne pojmove iz matrične algebre. Sastoji se od nekoliko koraka:

- priprema podataka
- računanje matrice kovarijanci
- računanje vlastitih vektora matrice kovarijanci
- odabrati prvih k vlastitih vektora i na kraju
- transformacija matrice ***X*** dimenzija *(n x m)* u matricu ***Z*** dimenzija *(k x m)*.

Zadatak 9. - Postupak ***analize glavnih komponenti*** ilustrirat ćemo koristeći podatke koje je 1996. g. objavio časopis ***Time Magazine***[100] koji povezuju očekivani vijek življenja i broj srčanih oboljenja na 100.000 stanovnika u jednoj godini uz konzumaciju broja litara žestokih alkoholnih pića, vina i piva u godini dana. Podaci su dani u tablici 6-8.

Tablica 6-8. Podaci s kojima ulazimo u PCA algoritam objavljeni u Times Magazine 1996. g.

Zemlja	Jaka alkoholna pića (l/g.)	Vino (l/g.)	Pivo (l/god)	Očekivani životni vijek (g.)	Srčane bolesti na 100000 st.
Francuska	2,5	63,5	40,1	78	61,1
Italija	0,9	58,0	25,1	78,0	94,1
Švicarska	1,7	46,0	65,0	78,0	106,4
Austrija	1,2	15,7	102,1	78,0	173,0
UK	1,5	12,2	100,0	77,0	199,7
USA	2,0	8,9	87,8	76,0	176,0
Rusija	3,8	2,7	17,1	69,0	373,6
Češka	1,0	1,7	140,0	73,0	283,7
Japan	2,1	1,0	55,0	79,0	34,7
Meksiko	0,8	0,2	50,4	73,0	36,4

PCA analizu na ovom skupu podataka proveli smo koristeći matematički programski jezik ***Octave***[101] koji već ima ugrađene funkcije koje nam značajno olakšavaju analizu. Slična je situacija i u ***Matlabu***.

Priprema podataka

Podatke kojima želimo smanjiti dimenzionalnost zapisujemo u obliku matrice ***X***, s tim da su pojedinačni uzorci skupa za treniranje pojedini redovi ove matrice. Ima ih ukupno m $(j = 1, ..., m)$. Stupaca ima n $(i = 1, ..., n)$ i svaki od njih je jedna značajka (dimenzija) uzorka. Sve značajke moraju imati brojčane vrijednosti. Pokazalo se kako su rezultati algoritma najbolji ako su sve značajke normirane i skalirane. Uobičajena metoda kod PCA naziva se standardizacija (engl. *Standardisation*) kod koje se značajke transformiraju u standardnu normalnu distribuciju kod koje je $\mu = 0$ i $s = 1$ jednadžbom:

$$x_{ij} = \frac{x_{ij} - \mu_i}{s_i} \tag{6-24}$$

gdje je x_{ij} podatak iz j-tog retka i-te značajke, μ_i je srednja vrijednost značajke (i-tog stupca matrice ***X***), a s_i standardna devijacija tog istog stupca. Konačni rezultat je normirana matrica $\boldsymbol{X_m}$.

U programu Octave to se postiže funkcijama:

```
mu = mean(X)
```

[100] Izvor podataka https://www.thedataschool.co.uk/robbin-vernooij/principal-component-analysis-alteryx-example-pinguin/. Ponovili smo primjer, ali koristeći Octave biblioteku.

[101] Octave se može skinuti sa stranice https://www.gnu.org/software/octave/, a može se koristiti i *online* verzija https://octave-online.net.

```
Xm = bsxfun(@minus, X, mu)
```

Rezultat je:

```
>> X = [2.5 63.5 40.1 78 61.1;0.9 58.0 25.1 78.0 94.1;1.7 46.0 65.0 78.0 106.4;
1.2 15.7 102.1 78.0 173.0;1.5 12.2 100.0 77.0 199.7;2.0 8.9 87.8 76.0 176.0;
3.8 2.7 17.1 69.0 373.6;1.0 1.7 140.0 73.0 283.7;2.1 1.0 55.0 79.0 34.7;
0.8 0.2 50.4 73.0 36.4];
>>
>> X
X =

     2.50000    63.50000    40.10000    78.00000    61.10000
     0.90000    58.00000    25.10000    78.00000    94.10000
     1.70000    46.00000    65.00000    78.00000   106.40000
     1.20000    15.70000   102.10000    78.00000   173.00000
     1.50000    12.20000   100.00000    77.00000   199.70000
     2.00000     8.90000    87.80000    76.00000   176.00000
     3.80000     2.70000    17.10000    69.00000   373.60000
     1.00000     1.70000   140.00000    73.00000   283.70000
     2.10000     1.00000    55.00000    79.00000    34.70000
     0.80000     0.20000    50.40000    73.00000    36.40000

>>
>>
>>
>> mu = mean(X)
mu =

     1.7500    20.9900    68.2600    75.9000   153.8700

>>
>> sd = std(X)
sd =

     0.91318    24.92702    38.67179     3.21282   110.81817
```

```
>>
>> Xm1 = bsxfun(@minus, X, mu);
>>
>> Xm = bsxfun(@rdivide, Xm1, sd)
Xm =

   0.821310   1.705378  -0.728179   0.653631  -0.837137
  -0.930818   1.484734  -1.116059   0.653631  -0.539352
  -0.054754   1.003329  -0.084299   0.653631  -0.428359
  -0.602294  -0.212219   0.875057   0.653631   0.172625
  -0.273770  -0.352629   0.820753   0.342378   0.413560
   0.273770  -0.485016   0.505278   0.031125   0.199696
   2.244914  -0.733742  -1.322928  -2.147645   1.982798
  -0.821310  -0.773859   1.855099  -0.902633   1.171559
   0.383278  -0.801941  -0.342886   0.964884  -1.075365
  -1.040326  -0.834035  -0.461835  -0.902633  -1.060025
```

Računanje matrice kovarijanci

Matrica kovarijanci (engl. *Covariance Matrix)* $\boldsymbol{\Sigma}$ je kvadratna matrica dimenzija *(n x n)* koja se izračuna jednadžbom:

$$\boldsymbol{\Sigma} = \frac{1}{m}(\boldsymbol{X}^T \cdot \boldsymbol{X}) \tag{6-25}$$

Matrica kovarijanci govori o korelaciji, povezanosti između varijabli, u našem slučaju između značajki. Ako joj je vrijednost nekog elementa σ_{jl} pozitivna, to znači da između značajki j i l postoji pozitivna korelacija, ako raste jedna, i druga će rasti i obrnuto. Za negativne vrijednosti elementa σ_{jl} povezanost je inverzna, kada jedna značajka raste, druga će padati.

U programu Octave matrica kovarijanci jednostavno se računa naredbom:

```
C = cov(Xm)
```

a rezultat je:

```
>> C = cov(Xm)
C =

   1.000000  -0.044005  -0.479160  -0.373038   0.428336
  -0.044005   1.000000  -0.389820   0.524422  -0.394727
  -0.479160  -0.389820   1.000000   0.095653   0.249731
  -0.373038   0.524422   0.095653   1.000000  -0.701743
   0.428336  -0.394727   0.249731  -0.701743   1.000000
```

Pronalazak vlastitog vektora i vlastite vrijednosti matrice kovarijanci

Sljedeći je korak određivanje ***vlastitog (svojstvenog) vektora*** (engl. *Eigenvector*) matrice i ***vlastite (svojstvene) vrijednosti*** (engl. *Eigenvalue*). Vlastiti vektor i vlastite vrijednosti matrice vezani su za svojstva kvadratne matrice koja ima jednak broj redaka i stupaca, a to je upravo naša matrica kovarijanci. Realni broj λ zove se vlastita vrijednost (ili karakteristična) matrice matrice $\boldsymbol{\Sigma}$ i piše se $\lambda=eig(\boldsymbol{\Sigma})$ onda i samo onda ako postoji vektor $\boldsymbol{x}$ takav da zadovoljava jednadžbu:

$$\boldsymbol{\lambda} \cdot \mathbf{x} - \boldsymbol{\Sigma} \cdot \mathbf{x} = (\boldsymbol{\lambda} \cdot \mathbf{I} - \boldsymbol{\Sigma}) \cdot \mathbf{x} = 0 \qquad (6\text{-}26)$$

$\boldsymbol{x}$ se naziva vlastiti vektor matrice $\boldsymbol{\Sigma}$ koji pripada vlastitoj vrijednosti λ. Jednadžba će imati netrivijalno rješenje samo u slučaju ako je $det(\boldsymbol{\lambda} \cdot \boldsymbol{I} - \boldsymbol{\Sigma}) = 0$, a rješenja ove jednadžbe daju i sve vlastite vrijednosti matrice $\boldsymbol{\Sigma}$. Nakon pronalaska vlastitih vrijednosti jednadžbom (6-26) proračunavamo i sve vlastite vektore. Ovaj je postupak dosta računski zahtjevan, te već i kod malo većih matrica zahtijeva korištenje naprednijih programskih alata. Matematički programski jezici tipa ***Octave*** i ***Matlab*** imaju dvije funkcije kojima se mogu računati vlastiti vektori i vlastite vrijednosti. Funkcija:

```
[V,D] = eig(C)
```

vraća matricu V u kojoj su vlastiti vektori i dijagonalnu matricu D u kojoj su vlastite vrijednosti. Jedini je problem što vlastiti vektori nisu sortirani od najvećeg prema najmanjem, pa treba naknadno napraviti i sortiranje:

```
[D, i] = sort(diag(D), 'descend')
V = V(:,i)
```

Konačan rezultat jest:

```
>> [V,D] = eig(C)
V =

  -0.329471   0.634911  -0.214391  -0.568092   0.345896
  -0.276418  -0.448112  -0.617713  -0.378363  -0.445039
  -0.496570   0.206755  -0.424780   0.724405   0.073961
   0.505694   0.567461  -0.269428   0.086423  -0.584980
   0.559226  -0.176985  -0.565188   0.043377   0.578467

D =

Diagonal Matrix

   0.086433          0          0          0          0
          0   0.422213          0          0          0
          0          0   0.584236          0          0
          0          0          0   1.605738          0
          0          0          0          0   2.301380

>>
```

```
>> [D, i] = sort(diag(D), 'descend');
>>
>> D
D =

   2.301380
   1.605738
   0.584236
   0.422213
   0.086433

>>
>> V = V(:,i)
V =

   0.345896  -0.568092  -0.214391   0.634911  -0.329471
  -0.445039  -0.378363  -0.617713  -0.448112  -0.276418
   0.073961   0.724405  -0.424780   0.206755  -0.496570
  -0.584980   0.086423  -0.269428   0.567461   0.505694
   0.578467   0.043377  -0.565188  -0.176985   0.559226
```

Matrica ***V*** sadrži vlastite vektore složene po vlastitim vrijednostima vektora ***D***.

Odabrati prvih k vlastitih vektora i transformacija matrice *X u Z*.

Kako bismo odredili koliko ćemo vlastitih vektora odabrati, izračunat ćemo i koliko pojedina vlastita vrijednost utječe na ukupnu varijancu:

```
cumsum(D) / sum(D
```

Rezultat je:

```
>> cumsum(D) / sum(D)
ans =

   0.46028
   0.78142
   0.89827
   0.98271
   1.00000
```

što znači da prve dvije glavne komponente utječu na 78,1% ukupne varijance, pa se ostale mogu odbaciti. Reduciranu matricu ***V*** na k = 2 stupca označavamo sa V_{reduce}, te formiramo novu matrica ***Z*** koja ima smanjenu dimenzionalnost matrice ***X*** s početnih *n* značajki na smanjenih *k* značajki:

$$Z = Xm \cdot V_{reduce} \tag{6-27}$$

U spomenutim matematičkim programskim jezicima ovu je transformaciju krajnje jednostavno napraviti. Npr. u jeziku Octave programski kod glasi:

```
Vreduce = V(:,1:2)

Z = Xm * Vreduce
```

Rezultati su nove reducirane matrice V_{reduce} i ***Z***:

```
>> Vreduce= V(:,1:2)
Vreduce =

   0.345896  -0.568092
  -0.445039  -0.378363
   0.073961   0.724405
  -0.584980   0.086423
   0.578467   0.043377
```

```
>> Z = Xm*Vreduce
Z =

  -1.39535  -1.61915
  -1.75964  -0.80836
  -1.10185  -0.37168
  -0.33167   1.12033
   0.16189   0.93103
   0.44523   0.40536
   3.40852  -2.05563
   1.40325   2.07603
  -0.72239  -0.12596
  -0.10799   0.44802
```

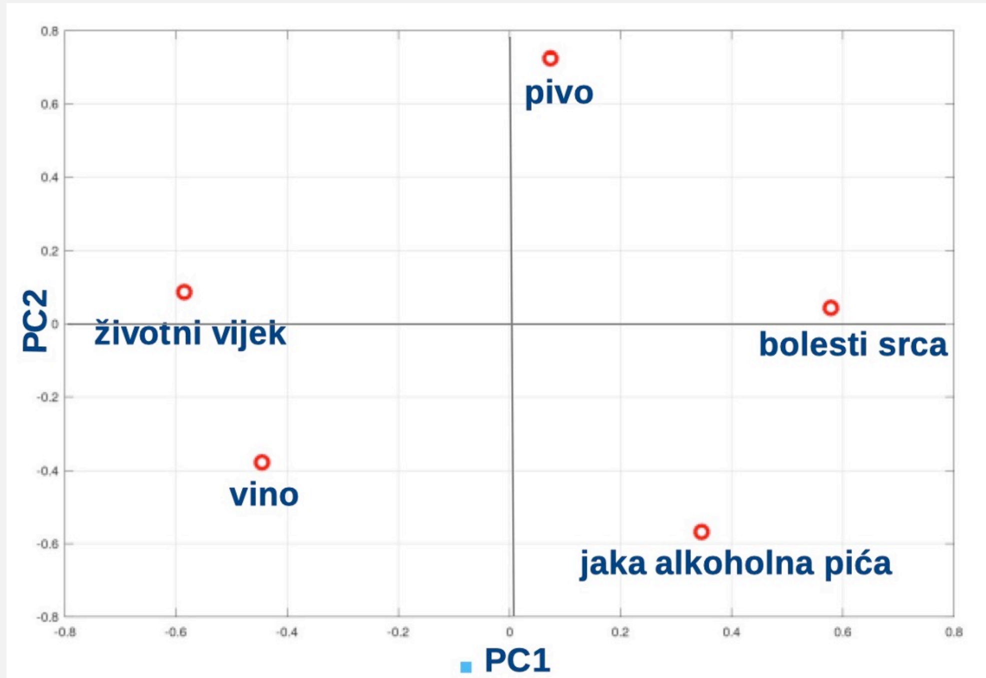
Što smo ovim dobili? Smanjili smo dimenzionalnost početne matrice koja je predstavljala Tablicu 6.8 i pokazivala za 10 zemalja 5 parametara vezanih uz konzumaciju alkohola i zdravlje. Tablicu 6-8 je vrlo teško grafički prikazati. Trebalo bi nam 5 osi koordinatnog sustava. PCA metodom smanjili smo dimenzionalnost na prve dvije glavne komponente PC1 i PC2 koje se sada mogu lako grafički prikazati ako PC1 i PC2 uzmemo za koordinatne osi. Na Slici 6-25 su dva prikaza.

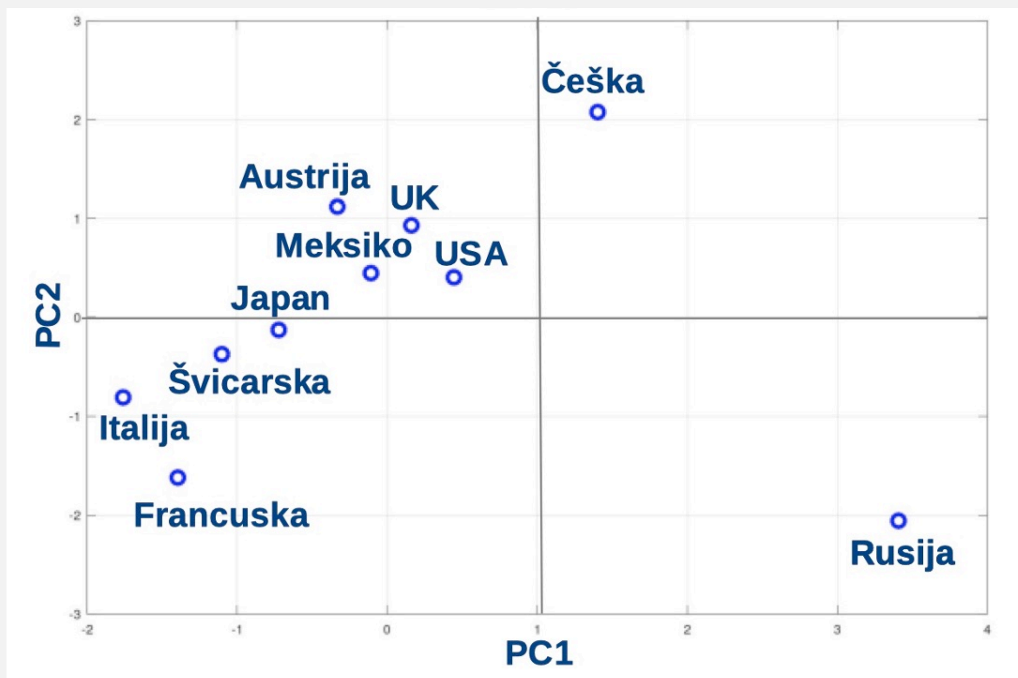
Jedan je položaj značajki u PC1-PC2 koordinatnom sustavu (matrica ***Vreduce***), a drugi položaj zemalja u PC1-PC2 koordinatnom sustavu (matrica ***Z***).

Iako su skale za značajke i zemlje različite, ako promatramo trendove s ova dva dijagrama puno toga možemo zaključiti što nismo mogli iz originalne tablice. Pogledajmo najprije značajke u odnosu na PC1:

- *Životni vijek i srčane bolesti su na suprotnim stranicama dijagrama što je i shvatljivo, u zemljama gdje ima više srčanih bolesti, manji je očekivani životni vijek.*

- *Konzumacija vina i životni vijek su međusobno blizu, što znači da se u zemljama gdje se pije vino duže živi.*
- *Konzumacija jakih alkoholnih pića i srčane bolesti su međusobno blizu, što znači da u zemljama gdje se piju jaka alkoholna pića ima više srčanih bolesti.*
- *Pivo je negdje u sredini u odnosu na životni vijek i srčane bolesti.*



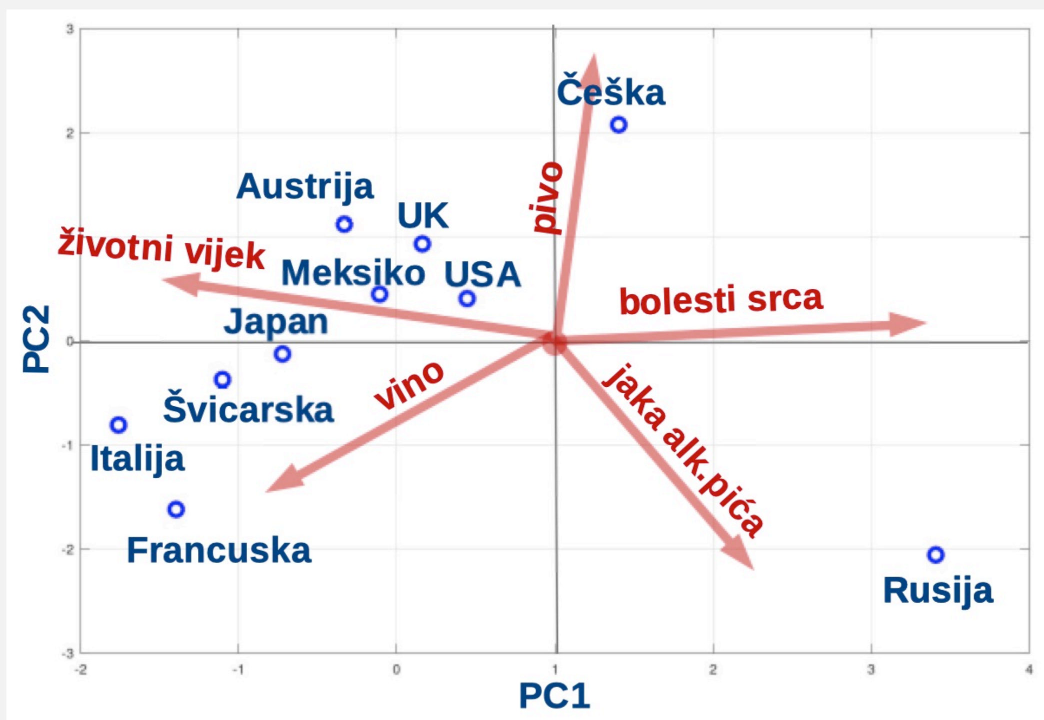


Slika 6-25. Prikaz značajki (gore) i zemalja (dolje) u koordinatnom sustavu određenom prvim dvjema osnovnim komponentama (PC1, PC2)

Pogledamo li distribuciju značajki u odnosu na PC2 os, jedino možemo zaključiti da je konzumacija jakih alkoholnih pića i vina na suprotnoj strani od konzumacije piva što znači da u zemljama u kojima se pije više jakih alkoholnih pića ili vina, pije se manje piva.

Pogledamo li raspored zemalja, prvo možemo primijetiti da su neke zemlje klasterirane što znači da su slične po konzumaciji pića, a neke odstupaju kao što su Češka, Rusija i Francuska. Ako na sliku zemalja ucrtamo smjerove značajki, dobijemo sliku 6-26 na kojoj možemo uočiti povezanosti između zemalja i značajki.

Tako je Češka najdalje u smjeru značajke konzumacija piva, Francuska i Italija u smjeru konzumacije vina, a Rusija u smjeru konzumacije jakih alkoholnih pića. To je bilo i za pretpostaviti, s tim da ovakve zaključke nije lako izvući iz tablice 6-8, ali je lako zaključiti sa slike 6-26.



Slika 6-26. Prikaz zemalja (dolje) u koordinatnom sustavu određenom prvim dvjema osnovnim komponentama (PC1, PC2) s ucrtanim smjerovima značajki sa slike značajki

Detekcija anomalija

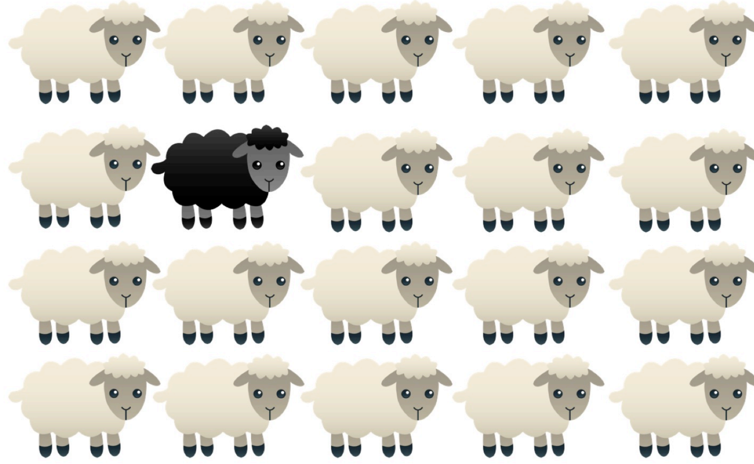
Detekcija anomalija (engl. *Anomaly Detection* ili *Outliner Detection*) je negdje između nadziranog i nenadziranog učenja, a temelji se na tome da imamo skup *„normalnih podataka"*, pa nastojimo ustanoviti je li se novi uzorak uklapa u ove *„normalne podatke"*. Ovaj postupak nalazi primjenu u brojnim područjima, od detekcija defektnog proizvoda na proizvodnoj traci u tvornici, detekcija sumnjivog ponašanja korisnika *online* bankarstva, detekcija kvara rada čvora u klasteru računala do prepoznavanje pogreški u radu programske podrške …

Detekcija anomalija spada u nadzirano učenje ako se klasifikator defektnih uzoraka trenira na temelju skupa podataka koji se smatraju *„normalnim"*. Ako novi uzorak nije sličan normalnim podacima, on se proglašava defektnim (anomalijom). Nenadzirano učenje je kada nemamo skup *„normalnih"* podataka, već u cijelom ulaznom skupu pokušavamo otkriti koji se uzorci razlikuju od većine ostalih uzoraka. Slika 6-27 ilustrira primjer gdje su *„normalni"* uzorci bijele ovce, zato što prevladavaju u skupu uzoraka, a anomalija je crna ovca koju bi algoritam detekcije anomalija trebao otkriti.

Jedan od postupaka nenadzirane detekcije anomalija temelji se na pretpostavci normalne distribucije uzoraka i koristi ***multivarijantnu statistiku*** (engl. *Multivariate Statistics*). Zbog toga se i naziva ***multivarijantna detekcija anomalija*** (engl. *Multivariate Anomaly Detection*).

Prvi je korak proračun vjerojatnosne distribucije $p(x)$. Za novi podatak provjerimo koliko se dobro uklapa u $p(x)$ svih podataka, te definiramo ***razinu granice*** (engl. *Threshold*) označene s ε na temelju

koje ćemo pojedini podatak proglasiti anomalijom. Ako je za neki uzorak x, $p(x) < \varepsilon$, proglašavamo ga anomalijom, a ako je $p(x) >= \varepsilon$, smatramo da se dobro uklapa u model i da je *„normalan”*.



Slika 6-27. Detekcija anomalija

Pri tome se najčešće koristite značajke Gaussove, normalne razdiobe. Pogledajmo primjer za slučaj kada se radi o podacima koji su opisani s više značajki. Prvi je korak odrediti koje bi značajke mogle biti indikativne za određivanje anomalija. Nakon toga za svih odabranih n značajki trebamo odrediti srednju vrijednost μ_j i standardnu devijaciju σ_j $(j=1, \dots , n)$ na temelju svih m uzoraka:

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \quad (6\text{-}28)$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2 \quad (6\text{-}29)$$

Za svaki novi uzorak x stupanj uklapanja u model izračunamo na temelju svih n odabranih značajki modela:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi\sigma_j}} e^{-\frac{(x_j-\mu_j)}{2\sigma_j^2}} \quad (6\text{-}30)$$

Uz odabranu granicu ε, ako je $p(x) < \varepsilon$, uzorak proglašavamo anomalijom. Postupak možemo ilustrativno objasniti na način da promatramo udaljenost novog uzorka od centra klastera kojem pripadaju *„normalni”* uzorci. Ako udaljenost prelazi neku granicu, uzorak smatramo anomalijom. Granica ε najčešće se računa na temelju faktora ***preciznosti*** (engl. *Precision*) i ***opoziva*** (engl. *Reccal*), koji se računaju iz dobro detektiranih uzoraka (*tp – true positive*), loše detektiranih uzoraka (*fp – false positive*) i loše detektiranih negativnih uzoraka (*fn – false negative*):

$$preciznost = tp/(tp + fp) \quad (6\text{-}31)$$

$$opoziv = tp/(tp + fn) \quad (6\text{-}32)$$

te na temelju njih izračunatog faktora kvalitete *F1*:

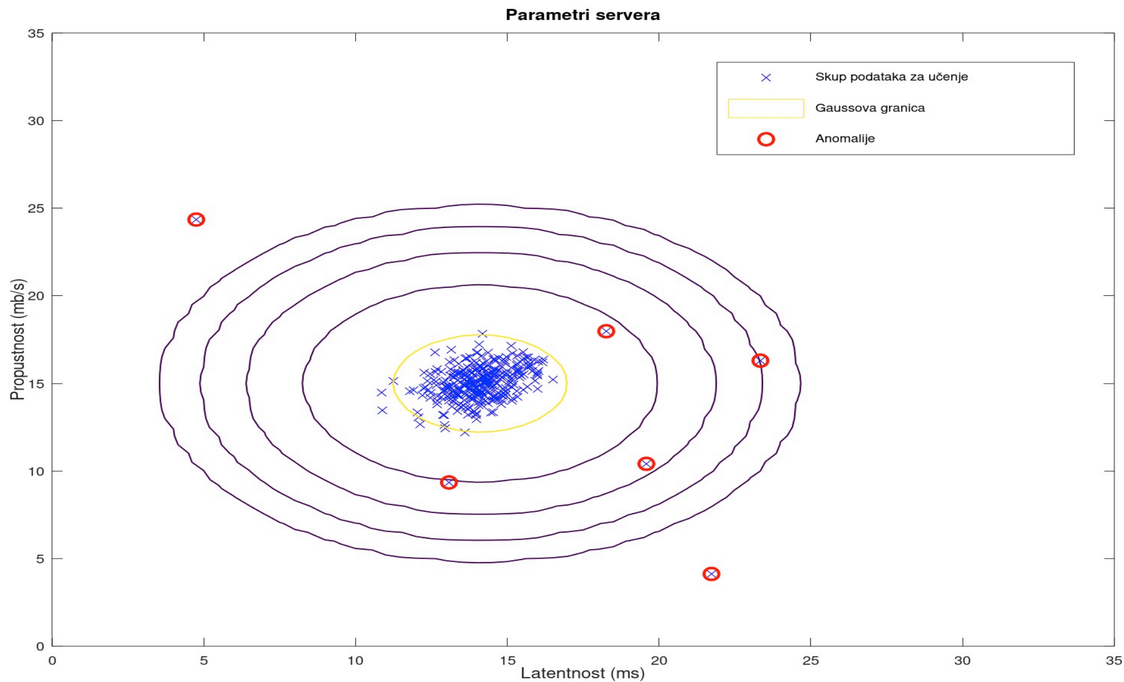
$$F1 = (2 \cdot preciznost \cdot opoziv)/(preciznost + opoziv) \quad (6\text{-}33)$$

Izabire se vrijednost granice ε za koju je *F1* najveći.

Pogledajmo jedan tipičan primjer kod kojeg imamo podatke o latentnosti servera u ms (latentnost servera je vrijeme koje serveru treba nakon slanja ping zahtjeva da odgovori na zahtjev) i propusnosti

servera u Mb/s (propusnost servera je količina podataka koja je u jedinici vremena uspješno prenesena sa servera)[102].

Na slici 6-28 prikazani su svi prikupljeni podaci, te je ucrtana odabrana granica. Kako su neki od podataka izvan ove granice, oni su prepoznati i označeni kao anomalije. Vrijednost parametra ε u ovom je primjeru bila $\varepsilon = 8{,}986095e\text{-}05$ za vrijednost faktora kvalitete $F1 = 0{,}8$.



Slika 6-28. Primjer automatske detekcije anomalija u radu servera. Na apscisnoj osi je latentnost servera u ms, a na ordinatnoj propusnost servera u Mb/s. Podaci koji su izvan odabrane Gaussove granice ε zaokruženi su crveno.

Nedostatak ove metode je što pretpostavlja ravnomjerni kružni raspored uzoraka oko centra klastera, pa na jednak način promatramo udaljenost uzorka u odnosu na sve značajke. Ako raspršenje uzoraka oko centra klastera nije kružno, tada je potrebno koristiti neke druge mjere udaljenosti.

6.5 Polunadzirano učenje

Polunadzirano učenje spada u hibridne metode koje imaju elemente i nadziranog i nenadziranog učenja. Ovdje ćemo se osvrnuti na dvije tehnike koje imaju veliku praktičnu primjenu:

- ***pojačano učenje*** i
- ***sustavi za davanje preporuka***.

6.5.1 Pojačano učenje

Pojačano učenje (engl. *RL – Reinforcement Learning)* temelji se na ideji nagrade i kazne. Za razliku od nadziranog učenja ovaj način učenja ne zahtijeva skup za treniranje. Sustav djeluje samostalno, bez ikakvih poznatih ulazno-izlaznih podataka, slično kao i sustavi nenadziranog učenja s jednom razlikom. Nakon svakog diskretnog ciklusa djelovanja sustav dobije ***nagradu*** (engl. *Reward*), pozitivnu ako je bio uspješan, a negativnu (kaznu) ako nije bio uspješan, što kod nenadziranog učenja

102 Primjer je prilagođen s https://github.com/trekhleb/machine-learning-octave/tree/master/anomaly-detection i riješen programom Octave.

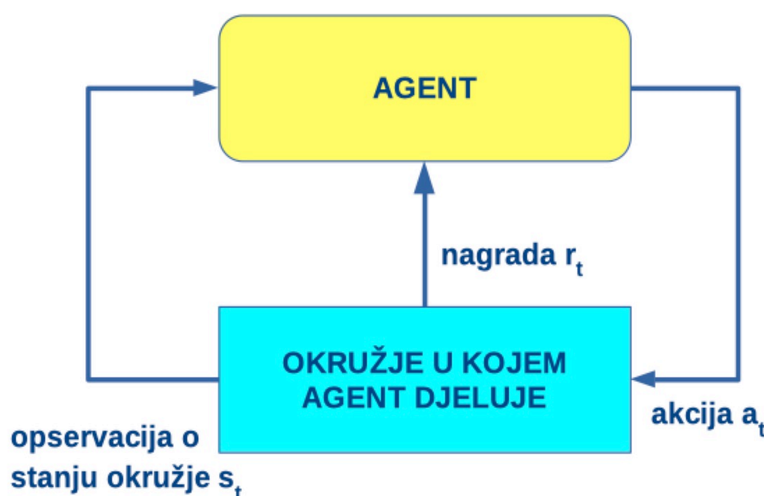
nemamo. Tipičan primjer iz života bi bio kada mala beba uči hodati i pokušava ustati. Kad se uspije održati na nogama, roditelji obično plješkaju i govore *„Bravo!"* što je za bebu ugoda i nagrada i zna da je na dobrom putu. S druge strane, ako nakon ustajanja padne, pad je na neki način kazna koja procesu učenja kaže da je nešto napravljeno pogrešno.

U poglavlju *3.6.2 Heuristično pretraživanje* upoznali smo Monte Carlo metodu pretraživanje stabla u igri dvaju igrača koja spada u pojačano učenje. Kod ove metode sljedeći potez u igri dvaju igrača birao se prema tome koja strategija najviše obećava, a tako je i kod pojačanog učenja.

Prve ideje o pojačanom učenju javljaju se još 60-ih godina 20. stoljeća i razvijaju se u tri pravca:

- učenje ***metodom pokušaj – pogriješi*** (engl. *Trial and Error*)
- rješavanje problema ***optimalnog vođenja*** (engl. *Optimal Control*), posebno korištenjem postupaka ***dinamičkog programiranja*** (engl. *Dynamic Programming*) i
- učenje metodama ***vremenskih razlika*** (engl. *TD – Temporal-Difference Methods*),

da bi koncem 80-ih godina 20. stoljeća konvergirali u suvremenu teoriju pojačanog učenja. Danas pojačano učenje nalazi veliku primjenu u robotici, posebno autonomnim robotskim vozilima, ali i u brojnim drugim područjima, na primjer kod upravljanja resursima u računalnim klasterima, upravljanja semaforima, konfiguriranja kompleksnih web-sustava s velikim brojem konfiguracijskih parametara, optimiranja kemijskih reakcija, podešavanja adaptivnog regulatora u sustavima automatskog vođenja itd. Shematski ga možemo prikazati slikom 6-29.



Slika 6-29. Shematski prikaz pojačanog učenja

Imamo agenta koji uči kako djelovati u nekom okružju. Agentova akcija a_i u diskretnom vremenskom trenutku t rezultat je analize osjetilnih informacija (opservacija) stanja okružja s_t i njegovog algoritma djelovanja A koji opservacije preslikava u akcije ($A: s_t \rightarrow a_t$). Kod pojačanog učenja agent nakon provedene akcije dobije i dodatni nagradni ulaz r_t koji mu kaže je li preslikavanje $s_t \rightarrow a_t$ bilo uspješno ili nije. Na temelju pojedinih nagradnih ulaza r_t agent formira kumulativnu nagradu pomoću koje mijenja algoritam djelovanja kako bi u sljedećim pokušaju bio uspješniji. Agent na početku ne zna koje su akcije dobre. On ih sam mora otkriti na temelju dobivenih nagrada.

Kod pojačanog učenja ***nagrada*** je skalarna povratna veza koja kaže koliko je agent bio uspješan u diskretnom trenutku t. Na temelju pojedinih nagrada postavlja se tzv. ***nagradna hipoteza*** (engl. *Reward Hypothesis*) koja kaže da nam je konačni cilj maksimiranje kumulativne nagrade koju računamo iz pojedinačnih nagrada. U okviru pojačanog učenja predložen je velik broj algoritama temeljenih na *dinamičkom programiranju*, *Monte Carlo metodi*, *procesima konačnih Markovljevih odluka* (engl. *Finite*

Markov Decision Processes) itd.[103], a mi ćemo ovdje detaljnije opisati dvije metode, metodu Q-učenja iz 1989. g. i metodu pojačanog slučajnog traženja iz 2018. g. koja na neki način vraća značaj perceptrona i jednostavnih umjetnih neuronskih mreža.

Q-učenje

Q-učenje (engl. *Q-learning*)[104] je postupak pojačanog učenja koje agentu sugerira koju će akciju primijeniti u nekoj situaciji. Spada u metode vremenskih razlika (engl. TD – *Temporal-Difference Methods*). Algoritam se temelji na učenju Q funkcije koja ocjenjuje koliko je dobra pojedina kombinacija opservacija – akcija ($s_t \rightarrow a_t$):

$$Q: S \; x \; A \; \rightarrow \; \mathbb{R} \tag{6-34}$$

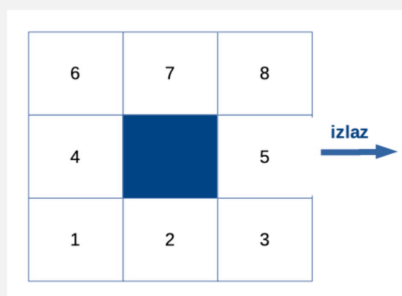
Slovo *Q* dolazi od engleske riječi *„quality"* koju prevodimo ***kvaliteta***, što znači da funkcija $Q(s_t,a_t)$ određuje kvalitetu određene poduzete akcije a_t za opservaciju s_t i to iskazuje realnim brojem. Njenu početnu vrijednost proizvoljno odabiremo, da bismo je u procesu učenja korigirali na temelju ostvarenih rezultata i primljenih nagrada. Pretpostavimo da smo za opservaciju s_t primijenili akciju a_t i dobili nagradu r_t, a djelovanje agenta rezultiralo je novom opservacijom s_{t+1}. Ova nova opservacija koja nam opisuje sadašnje stanje okružja rezultat je prethodnog stanja okružja s_t i primijenjene akcije a_t. Sljedeći je korak ispravak funkcije $Q(s_t,a_t)$ iteracijskim algoritmom:

$$Q^{NOVI}(s_t, a_t) = (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot \max_a Q(s_{t+1}, a)) \tag{6-35}$$

gdje je $\max_a Q(s_{t+1}, a)$ estimacija optimalnih budućih vrijednosti, α koeficijent brzine učenja $(0 < \alpha \leq 1)$, a γ također faktor između *0* i *1* $(0 \leq \gamma \leq 1)$ kojim se određuje koliki će se značaj dati estimaciji optimalnih budućih vrijednosti u odnosu na sadašnju nagradu r_t. Faktor *g* može biti i *0* što znači da se u obzir uzima samo sadašnja nagrada r_t. Q-učenje može biti epizodno kada postupak zaustavljamo ako je opservacija s_{t+1} konačna (završna) ili ne-epizodno kada se učenje nikada ne prekida.

Q-učenje ćemo ilustrirati jednostavnim primjerom traženja puta kroz jednostavni labirint prikazan na Slici 6-30.

Zadatak 10. – Primjer Q - učenja



Slika 6-30. Jednostavni labirint za ilustraciju postupka Q-učenja

Agent se nalazi na početnom položaju i ne zna gdje je izlaz. Svako polje unutar labirinta predstavlja jedno stanje, tako da su agentove opservacije brojčane oznake polja. U nekim je poljima prepreka, pa se agent u tim poljima ne može naći. Agent se može kretati samo horizontalno ili vertikalno. Sada uvodimo i nagradu. Ako agent udari u zid, nagrada je -1, ako može slobodno proći, nagrada je 0, a ako pronađe izlaz, nagrada je 100. Da bismo mogli primijeniti jednadžbu učenja, moramo poznavati vrijednost nagrade za bilo koju kombinaciju prelaska iz početnog stanja u konačno stanje.

[103] Za više detalja pogledati (Sutton and Barto, 2017).

[104] Q-učenja je 1989. g. predložio ***Chris Watkins*** u svojoj doktorskoj disertaciji *„Learning from Delayed Rewards"* obranjenoj na Cambridge Universityju, a 1992. objavio u radu (Watkins and Dayan, 1992).

Prikazujemo je matricom $\boldsymbol{R}$ veličine 8 x 8 zato što imamo osam polja po kojima se možemo kretati. Redci odgovaraju početnom stanju, a stupci sljedećem stanju. Usvojili smo oznaku -1 i za sva polja do kojih se ne može doći iz početnog stanja. Na primjer, pretpostavimo da krećemo iz polja (1). Iz njega su nam dostupna samo polja (2) i (4), a kako ona nisu izlazna polja, nagrada za njih je 0. Sva ostala polja su nedostupna, pa je nagrada -1.

$$\boldsymbol{R} = \begin{array}{c} \\ 1\\2\\3\\4\\5\\6\\7\\8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} -1 & 0 & -1 & 0 & -1 & -1 & -1 & -1 \\ 0 & -1 & 0 & -1 & -1 & -1 & -1 & -1 \\ -1 & 0 & -1 & -1 & 100 & -1 & -1 & -1 \\ 0 & -1 & -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & -1 & 0 & -1 & -1 & -1 & -1 & 0 \\ -1 & -1 & -1 & 0 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & -1 & -1 & 0 & -1 & 0 \\ -1 & -1 & -1 & -1 & 100 & -1 & 0 & -1 \end{bmatrix} \end{array}$$

Sljedeći je korak računanje matrice $\boldsymbol{Q}$ jednadžbom (6-29). Odabrat ćemo najveći koeficijent brzine učenja $\alpha = 1$ i faktor $\gamma = 0{,}8$, pa jednadžba (6-29) u tom slučaju glasi:

$$Q^{NOVI}(s_t, a_t) = r_t + 0{,}8 \cdot \max_a Q(s_{t+1}, a)$$

Početna $\boldsymbol{Q}$ matrica je nul matrica (svi elementi su nule), a nakon prve epizode (prvog ciklusa treniranja) dobijemo:

$$\boldsymbol{Q} = \begin{array}{c} \\ 1\\2\\3\\4\\5\\6\\7\\8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

zato što je:

$$Q^{NOVI}(3,5) = 100 + 0{,}8 \cdot \max_a[Q(5,8), Q(5,3)] = 100 + 0{,}8 \cdot 0 = 100$$

U drugom dijelu izraza uzimamo u obzir sve poteze koji se mogu dogoditi u budućnosti kada stignemo u polje (5). Iz njega možemo otići natrag u (8) i u polje (3). Tako računamo u svakoj sljedećoj epizodi. Nakon druge epizode promijenit će se još:

$$Q^{NOVI}(2,3) = 0 + 0{,}8 \cdot \max_a[Q(3,5), Q(3,2)] = 0{,}8 \cdot \max_a[100{,}0] = 80$$

$$Q^{NOVI}(7,8) = 0 + 0{,}8 \cdot \max_a[Q(8,5), Q(8,7)] = 0{,}8 \cdot \max_a[100{,}0] = 80$$

Nastavljamo dalje sve dok se matrica $\boldsymbol{Q}$ prestane mijenjati. Njen konačni oblik je:

$$\boldsymbol{Q} = \begin{array}{c} \\ 1\\2\\3\\4\\5\\6\\7\\8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} 0 & 64 & 0 & 40{,}96 & 0 & 0 & 0 & 0 \\ 51{,}2 & 0 & 80 & 0 & 0 & 0 & 0 & 0 \\ 0 & 64 & 0 & 0 & 100 & 0 & 0 & 0 \\ 51{,}2 & 0 & 0 & 0 & 0 & 51{,}2 & 0 & 0 \\ 0 & 0 & 80 & 0 & 0 & 0 & 0 & 80 \\ 0 & 0 & 0 & 40{,}96 & 0 & 0 & 64 & 0 \\ 0 & 0 & 0 & 0 & 0 & 37{,}77 & 0 & 80 \\ 0 & 0 & 0 & 0 & 100 & 0 & 26{,}21 & 0 \end{bmatrix} \end{array}$$

Ova konačna matrica kvalitete daje odgovore kamo krenuti iz bilo kojeg početnog položaja kako bismo najprije došli do izlaza koji je polje(5). Kada smo došli do polja(5), zadatak je riješen. Naš je izbor uvijek onaj koji ima maksimalnu vrijednost u matrici $\boldsymbol{Q}$.

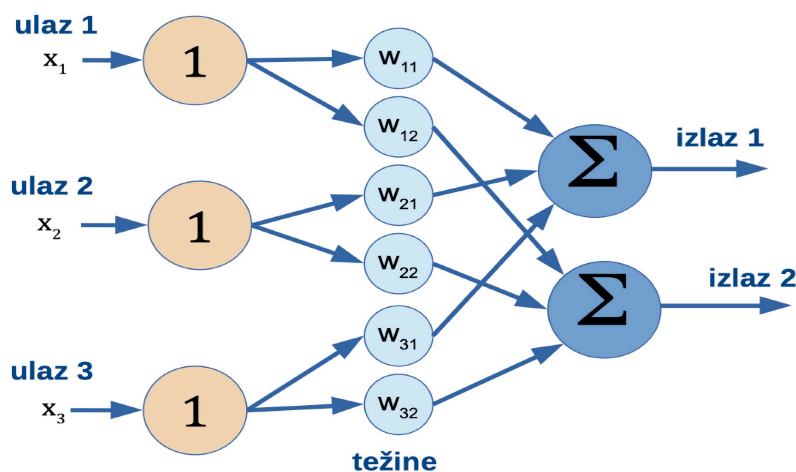
Na primjer, krećemo iz polja (1). U prvom retku matrice $\boldsymbol{Q}$ vidimo da imamo dva izbora – polje (2) s indeksom kvalitete 64 i polje (4) s indeksom kvalitete 40,96. Idemo u polje (2). Odavde nam se nudi polje (1) s indeksom kvalitete 31,2 i polje (3) s indeksom kvalitete 80. Idemo u polje (3) iz kojeg

možemo natrag u (2) s indeksom 64 i u polje (5) s indeksom 100 koje je i naš cilj. Zanimljiva je i situacija kada krećemo iz (4). Imamo dvije opcije iste kvalitete (1) i (6), pa je u algoritam potrebno ugraditi dodatni kriterij kojim se razrješava ovakva dilema. Primjer je *„uzmi prvo rješenje na koje si naišao"*.

U ovom smo primjeru na početku imali cijelu matricu ***R***. A što ako je nemamo osim što znamo da je nagrada 0 ako nam je neko polje dostupno i konačna nagrada 100 kada dođemo do izlaza? U tom slučaju trebamo sami izgraditi matricu ***R*** na temelju koje kasnije proračunavamo matricu ***Q***. Kako? Tako da istražujemo. Na primjer, nalazimo se u polju (1). Ispitamo sve moguće smjerove i lako otkrijemo da je moguće samo kretanje *„lijevo"* kojim dolazimo u (2) i *„gore"* kojim dolazimo u (4). To nam je dovoljno za ispunjavanje prvog retka matrice ***R***. Nastavljamo dalje dok ne popunimo cijelu matricu iz koje onda izračunamo matricu ***Q*** koja nam daje rješenje za sve moguće situacije u kojima bismo se mogli pronaći. To je postupak učenja. Primjer iz stvarnog života bi mogao biti robot dostavljač koji unutar neke firme treba dostavljati pošiljke do pojedinih zaposlenih. On najprije uči o prostoru u kojem se nalazi, na primjer koristeći opisani postupak Q-učenja. Nakon toga robot može rutinski obavljati posao odabirući uvijek najkraći put do svog odredišta.

Pojačano slučajno traženje

Pojačano slučajno traženje (engl. *ARS – Augmented Random Search*)[105] je način učenja kojem je za cilj pronaći postupak koji će maksimirati nagradu koju bi agent mogao dobiti ako u danom okružju prati baš taj postupak. ARS je je unaprijeđeni postupak **osnovnog slučajnog traženja** (engl. *BRS – Basic Random Search*) koji ćemo ilustrirati na učenju težina jednostavne umjetne neuronske mreže bez međuslojeva prikazane na slici 6-31.



Slika 6-31. Jednostavna umjetna neuronska mreža na kojoj ćemo ilustrirati postupak osnovnog i pojačanog slučajnog traženja

Ovakvu mrežu uobičajeno nazivamo „***plitka***" (engl. *Shallow*) mreža, a postupak ***„plitko učenje***" (engl. *Shallow Learning*) za razliku od dubokih umjetnih neuronskih mreža koje imaju brojne međuslojeve. Mreža ima samo dva sloja, ulazni i izlazni, tri ulazne vrijednosti i dvije izlazne vrijednosti, pa je ukupno šest parametara za podešavanje w_{ij}. Nakon svake primijenjene akcije sustav primi odgovarajuću ***„nagradu"*** *R* kako je i uobičajeno kod pojačanog učenja.

Ideja osnovnog slučajnog traženja je da se krene od nekih početnih parametara, te se naprave dva pokusa. Kod jednog se promijene za slučajno odabranu pozitivnu malu vrijednost $+d_{ij}$, a kod druge za istu

[105] ARS postupak su 2018. g. predložili ***Horia Mania, Aurelia Guy i Benjamin Recht*** u radu (Mania, Guy, Recht, 2018.)

tu, ali negativnu vrijednost $-d$ ($w_{ij}^{PLUS} = w_{ij} + d$, $w_{ij}^{MINUS} = w_{ij} - d$). Za ova dva slučaja dobit će se dvije nagrade R_{PLUS} i R_{MINUS}, na temelju kojih se računaju novi parametri w_{ij} :

$$w_{ij}^{NOVI} = w_{ij} + \alpha \cdot \Delta = w_{ij} + \alpha \cdot (R_{PLUS} - R_{MINUS}) \cdot d \qquad (6\text{-}35)$$

gdje je α uobičajeni koeficijent brzine učenja ($0 < \alpha \leq 1$).

U jednom koraku učenja uobičajeno je napraviti više ovakvih perturbacija, na primjer N, pa se tek nakon svih izvršenih N testiranja korigiraju parametri. Ovdje trebamo napomenuti da se u biti napravi $2N$ testiranja, zato što se za svaki d_k provodi testiranje i za pozitivnu i za negativnu promjenu. U tom se slučaju kao korekcijski faktor uzme srednja vrijednost razlike pozitivnih i negativnih nagrada:

$$w_{ij}^{NOVI} = w_{ij} + \frac{\alpha}{N} \cdot \sum_{k=1}^{N} (R_{PLUS}^{k} - R_{MINUS}^{k}) \cdot d_k \qquad (6\text{-}36)$$

Pojačano slučajno traženje uvodi još tri dodatna poboljšanja koja su značajno ubrzala postupak učenja:

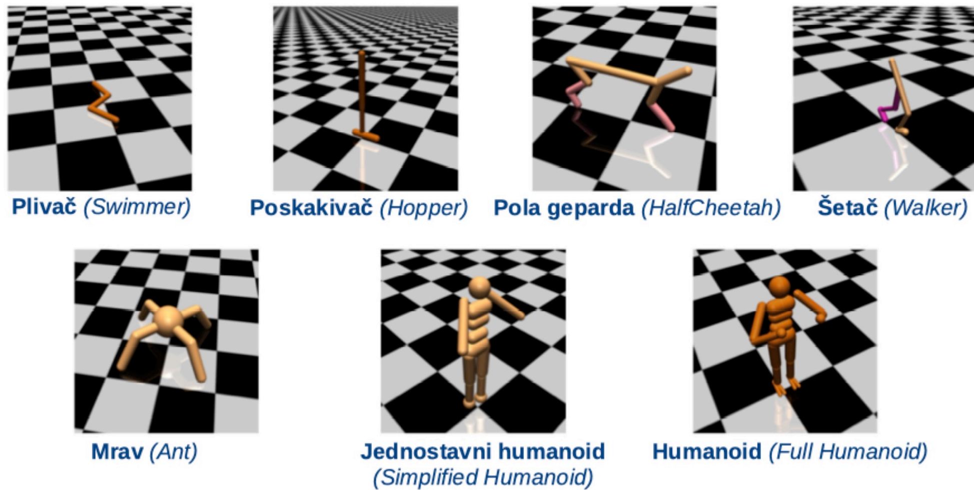
- dijeljenje korekcijskog parametra Δ sa standardnom devijacijom nagrada σ_R
- normalizacija svih ulaza i
- eliminacija onih testiranja koja su ostvarila najmanju nagradu.

Prvim poboljšanjem sprječava se veliko osciliranje korekcijskog faktora koje se zna pojaviti kod osnovnog slučajnog traženja. Drugim poboljšanjem ujednačavaju se ulazi ako poprimaju vrijednost u različitom skupu vrijednosti. Normalizacija se provodi jednostavno:

(ulazna_vrijednost – srednja_vrijednost_ulaza) /standardna_devijacija_ulaza

Trećim poboljšanjem eliminiraju se ona testiranja koja su rezultirala najmanjom nagradom pa se u jednadžbi (6-29) umjesto N testiranja izabere samo b najboljih.

Autori pojačanog slučajnog traženja (Mania, Guy, Recht, 2018.) algoritam su testirali na problemu koji se u posljednje vrijeme pojavljuje kao standardni referentni zadatak (engl. *Benchmark*) kod treniranja i testiranja algoritama strojnog učenja. Radi se o tzv. ***MuJoCo*** (engl. ***Mu****lti-***Jo****int dynamics with* ***Co****ntact)* virtualnoj platformi[106] koja nudi modeliranje, simulaciju i vizualizaciju dinamičkih modela različitog stupnja kompleksnosti s povratnom kontaktnom vezom tako da se može lako koristiti kod treniranja i testiranja algoritama za učenje kretanja (lokomocije) sustava različitog stupnja složenosti. Neke od njih prikazuje slika 6-32.



Slika 6-32. Strukture različite kompleksnosti tzv. ***MuJoCo*** *(****Mu****lti-Joint dynamics with* ***Co****ntact) sustava koje služe kao referentni zadaci kod treniranja i testiranja algoritama pojačanog učenja*

[106] Više detalja na http://www.mujoco.org

Jednostavni humanoid ima 17 upravljivih zglobova i 24 stupnja slobode gibanja (engl. *DoF – Degree of Freedom*), dok potpuni humanoid ima 21 upravljivi zglob i čak 28 stupnjeva slobode gibanja, pa oba predstavljaju teže zadatke učenja lokomocije.

Autori (Mania, Guy, Recht, 2018.) su testirali ovaj jednostavni algoritam pojačanog slučajnog učenja (ARS) na skoro svim lokomocijskim zadacima osim punog humanoida i usporedili s brojnim do tada poznatim algoritmima. Dobiveni rezultati pokazali su da je za iste postignute rezultate ARS algoritam 15 puta efikasniji od svih ostalih do tada predloženih algoritama.

6.5.2 Davanje preporuka

Sustavi za davanje preporuka (engl. *Recommender Systems)* nisu nova tehnologija, već primjena postojećih i već opisanih tehnologija. Možemo ih svrstati u polunadzirano učenje zato što postupak ovisi o ocjeni korisnika. Posljednjih godina dosta ih koriste velike *online* platforme kao što su Amazon, Facebook, eBay, Pandora itd.

Sustavi za preporuku s jedne strane uče model korisnika kroz njegove aktivnosti, a s druge strane uče model proizvoda ili usluge na temelju interakcija s poznatim, već modeliranim korisnicima. Nakon toga sustavi sparuju korisnike i proizvode temeljem naučenih modela, te korisnicima nude one proizvode za koje pretpostavljaju da će se baš njima svidjeti. Drugim riječima kazano, sustavi za preporuku stvaraju listu preporuka korisnicima temeljem njihovih prethodnih aktivnosti i aktivnosti drugih korisnika.

Pogledajmo ukratko kako bi mogao funkcionirati ovakav sustav na primjeru ***sustava za preporuku filmova***.

Zadatak 11 – Primjer zadatka davanja preporuka

Zadatak mu je da registriranim korisnicima preporučuje listu filmova za gledanje. Kako ljudi imaju različite ukuse, kako bismo nekome uspješno preporučili film moramo poznavati značajke te osobe vezane uz filmove, na primjer je li radije gleda komedije, drame ili romantične sadržaje, voli li specijalne efekte i slično, ali isto tako treba imati znanje o filmovima koje preporučuje. Važan dio ovakvih sustava je to što registrirani korisnici imaju mogućnost ocjenjivanja i recenzije filmova koje su pogledali. Njihove ocjene daju drugim korisnicima i sustavu informaciju o filmu, ali i sustavu daju uvid u naklonosti korisnika. Neke od ocjena korisnika su u tablici 6-9:

Tablica 6-9. Filmovi ocijenjeni od strane korisnika i procjene značajke filmova u odnosu na „romantiku" i „akciju" od strane eksperta

Film	*Ana*	*Ivan*	*Eva*	*Tin*	*romantika (x1)*	*akcija (x2)*
Za ljubav nema lijeka	5	5	0	0	0,9	0
Najduže putovanje	5	?	?	0	1,0	0,01
Hvala ti za ljubav	?	4	0	?	0,99	0
Neka bolji pobijedi	0	0	5	4	0,1	1,0
Posljednja patrola	0	0	5	?	0	0,9

Korisnici Ana, Ivan, Eva i Tin različito su ocijenili iste filmove. Npr. film „Za ljubav nema lijeka" ima neke značajke koje se sviđaju Ani i Ivanu, ali se Evi i Tinu ne sviđa. Film „Nonstop car chases" sadrži značajke koje se sviđaju Evi i Tinu, ali se Ani i Ivanu ne sviđaju. Dobar poznavalac filmova (ekspert) ručno pridjeljuje pojedinim filmovima vrijednost različitih značajki u intervalu od 0 do 1. U ovom primjeru uzet ćemo dvije značajke „romantika" (x_1) i „akcija" (x_2). Na primjer, ekspert je za film „Za ljubav nema lijeka" procijenio da je izuzetno romantičan bez akcije, pa mu je dao

vrijednosti x1 = 0,9 i x2 = 0. Na temelju ovih podataka gradimo modele korisnika, na primjer Ane. Postupak koji ćemo primijeniti je linearna regresija s dvije varijable. Za nju nam treba matrica X koju dobijemo da značajkama dodajemo i prvi stupac x_0 s jedinicama. Kako Ana nije ocijenila treći film, taj redak izostavljamo. Matrična jednadžba linearne regresije s dvije varijable i njeno rješenje je[107]:

$$\begin{bmatrix} 1 & 0,9 & 0 \\ 1 & 1 & 0,01 \\ 1 & 0,1 & 1 \\ 1 & 0 & 0,9 \end{bmatrix} \cdot \boldsymbol{\Theta} = \begin{bmatrix} 5 \\ 5 \\ 0 \\ 0 \end{bmatrix} \quad ; \quad \boldsymbol{\Theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 2,948 \\ 2,175 \\ -3,215 \end{bmatrix}$$

pa regresijski model Ane glasi:

ocjena_Ana = 2,948 + 2,175· romantika – 3,2148 · akcija

Sada kada imamo model korisnika, možemo predvidjeti njegove ocjene za filmove koje još nije ocijenio. Iako Ana nije ocijenila film „Hvala ti za ljubav", temeljem modela možemo predvidjeti da bi se njoj svidio, tj. da će mu dati visoku ocjenu:

ocjena_Ana = 2,948 + 2,175· 0,99 – 3,2148 · 0 = 5 (zaokruženo na cijeli broj)

Na isti način možemo kreirati modele ostalih korisnika te procijeniti hoće li im se pojedini filmovi svidjeti ili ne.

Problem je što nije lako pronaći eksperta, te značajke filma unositi ručno. Ako imamo ocjene nekoliko korisnika čiji model poznajemo, možemo procijeniti značajke filmova koristeći te informacije korisnika i modelirati filmove pomoću informacije kojim su se korisnicima sviđali. Ako je jedan korisnik čiji je model definiran s θ_1 dao filmu nepoznatih značajki ocjenu 5, a korisnik modela θ_2 dao ocjenu 0, značajke filma možemo izračunati iz jednadžbi:

ocjena_1 = $\boldsymbol{\theta_{10}} + \boldsymbol{\theta_{11}} \cdot$ *romantika* + $\boldsymbol{\theta_{12}} \cdot$ *akcija*

ocjena_2 = $\boldsymbol{\theta_{20}} + \boldsymbol{\theta_{21}} \cdot$ *romantika* + $\boldsymbol{\theta_{22}} \cdot$ *akcija*

što matrično možemo napisati:

$$\begin{bmatrix} \Theta_1^T \\ \Theta_2^T \end{bmatrix} \cdot \begin{bmatrix} 1 \\ romantika \\ akcija \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

U našem primjeru modeli Ivana i Eve su:

$$\boldsymbol{\Theta_1} = \begin{bmatrix} \theta_{10} \\ \theta_{11} \\ \theta_{12} \end{bmatrix} = \begin{bmatrix} 4,457 \\ 0,033 \\ -4,680 \end{bmatrix} \quad \boldsymbol{\Theta_2} = \begin{bmatrix} \theta_{20} \\ \theta_{21} \\ \theta_{22} \end{bmatrix} = \begin{bmatrix} 2,109 \\ -2,229 \\ 3,158 \end{bmatrix}$$

pa su značajke za ovaj film dobivene na temelju ocjena Ivana i Eve: romantika = 0,786 i akcija = -0,113. Ako ostanemo na tome da ocjene značajki trebaju biti u intervalu između 0 i 1, onda bismo akciju promijenili u 0.

....

Dodatak: Dobar primjer korištenja strojnog učenja u umjetnosti je slika ***„Djevojka iz Trsta"*** (tal. *La Ragazza di Trieste*) koja predstavlja portret talijanske glumice ***Ornelle Mutti***, filmske zvijezde iz 80-ih godina 20. stoljeća u stilu slike *„Fornarina"* talijanskog renesansnog slikara ***Rafaela***. Sliku je *„nacrtao"* multimedijski umjetnik ***Joseph Aerle*** koristeći posebnu duboku neuronsku mrežu koja se zove ***generička suparnička mreža*** (engl. *GANs - Generative Adversariel Networks*). Princip rada generičke suparničke mreže prikazuje slika ispod. GAN mreža se sastoji od dvije umjetne neuronske mreže, koje se međusobno natječu, kako bi stvorile nove generičke instance kojima se mogu zamijeniti stvarni podatci.

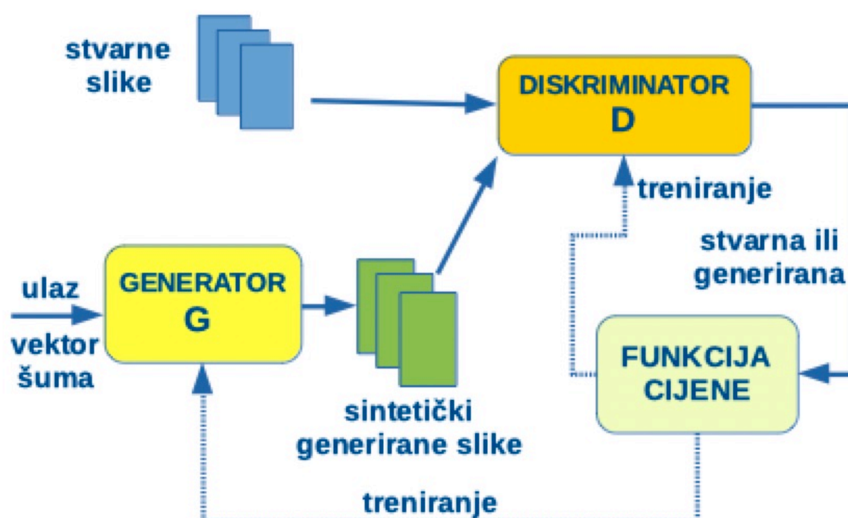
[107] Koristili smo isti *online* kalkulator
https://home.ubalt.edu/ntsbarsh/Business-stat/otherapplets/MultRgression.htm



Umjetno generirana slika Ornelle Mutti.

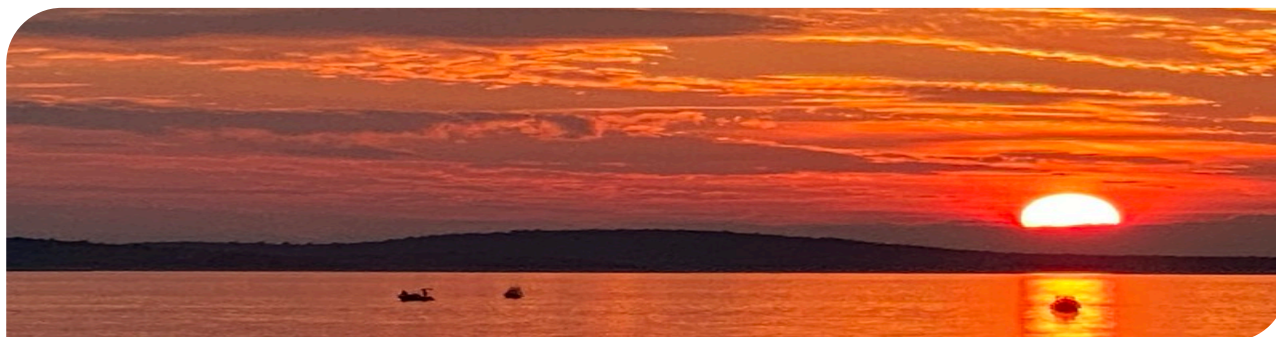


Originalna slika Rafaela „Fornarina"



Prva mreža se naziva se ***generator G***, a njen je zadatak generirati umjetnu, sintetički generiranu sliku. Druga mreža naziva se ***diskriminator D***, koji pokušava otkriti da li se radi o stvarnoj ili generiranoj slici. Generator nema pristup realnim podacima, njegov učitelj je diskriminator. Diskriminator ima pristup i sintetički generiranim podacima i stvarnim podacima. Njegov zadatak je razlučiti da li je podatak koji je generiran od strane generatora stvarni podatak ili sintetički generirani podatak. Mreže su obično implementirane koristeći višeslojne arhitekture. Treniranje GAN mreže je završeno kada diskriminator *„odluči"* da sintetički generirane slike dobro *„odgovaraju"* realnim slikama. Mreža je trenirana na originalnoj slici ***Raffaela*** koristeći njegove značajke boje i stila, a nakon toga je generator ulaznu sliku ***Ornelle Mutti*** preradio u stilu ***Rafaella***. Aerle je napravio i film s istom glumicom naslova ***„Emocije zauvijek"*** (tal. *Un'emozione per sempre 2.0*) generirajući iz ***Ornelle Mutti*** sintetičku glumicu. Tehnika se zove ***„Deepfake"***, a naziv je nastao kombinirajući riječi ***„Deep learning"*** (hrv. *duboko učenje*) i ***„Fake"*** (hrv. *lažno*). Više detalja na https://www.wikiwand.com/en/Deepfake , a radove umjetnika, uključujući i cjelovitu sliku možete pronaći na https://www.saatchiart.com/account/artworks/854436 .

POGOVOR



”

Teme obrađene u ovom udžbeniku daju tek kratki uvod u područje nebioloških inteligencija i inteligentnih tehnologija koje su u posljednjem desetljeću postale sastavni dio našeg života, pa se sve češće čuju pojmovi kao što su **inteligentna vozila**, **inteligentne prometnice**, **inteligentne kuće**, pa čak i **inteligentna odjeća**. U budućnosti će toga biti još više, pa smo zbog toga ovim udžbenikom čitateljima nastojali bar malo približiti ovo zanimljivo područje.

Ovaj se udžbenik zove *Uvod u umjetnu inteligenciju*, ali nam se čini da je čak i to previše ambiciozno. Pravi bi naslov možda trebao biti *Kratki uvod u umjetnu inteligenciju*. ***Stuard C. Shapiro*** je 1992.g. bio urednik drugog izdanja dvovolumne knjige „*Encyclopedia of Artificial Intelligence”* (Shapiro, 1992.) koja je imala 1724 stranica. To je bilo 1992. g., a u proteklih 30 godina znanje iz područja inteligentnih tehnologija nekoliko puta povećalo, dok ovaj udžbenik ima samo 320 stranica. Nastojali smo uključiti ono znanje za koje smatramo da je bitno za razumijevanje ovog područja, koje već sada bitno utječe na naše živote, a u budućnosti će, sigurni smo, utjecati još i više. Inteligentna vozila, inteligentne prometnice, inteligentne kuće, inteligentna odjeća, inteligentni kućanski aparati, samo su neke od primjena inteligentnih tehnologija u našem svakodnevnom životu kojih će u sljedećim godinama biti sve više i više. Sve su ove primjeri primjene ***inteligentnih sustava***. Inteligentni sustavi samostalno djeluju u svom okružju i imaju tri osnovna dijela:

- osjetila ili senzore kojima opažaju svijet oko sebe
- izvršne uređaje ili aktuatore kojima povratno djeluju na svijet koji ih okružuje i
- centralnu jedinicu u kojoj se obrađuju i interpretiraju osjetilne informacije, provodi zaključivanje i donose odluke o povratnom djelovanju na okružje u kojem inteligentni sustav djeluje i to prije svega na temelju pohranjenog znanja.

U ovom smo se udžbeniku bavili samo trećim dijelom i vrlo kratko pokazali osnovne metode rješavanja kompleksnijih zadataka na temelju prikupljenog, kodiranog i pohranjenog znanja. Nismo se bavili osjetilima, interpretacijom osjetilnih informacija, izvršnim uređajima i generiranjem njihovih upravljačkih signala. Potičemo čitatelje da sami krenu u istraživanje ovih izazovnih i uzbudljivih područja. Razumijevanje teksta, sažimanje teksta, razumijevanje slike, proaktivno ponašanje, planiranje, smišljeno vođenje, samo su neka od tih područja koja će sigurno obilježiti godine koje su ispred nas.

LITERATURA





Autori ovog udžbenika već više od 20 godina predaju različite varijante kolegija Umjetna inteligencija. Pri tome su došli u kontakt sa stotinama knjiga i časopisa na temu umjetne inteligencije i inteligentnih tehnologija. U ovom kratkom popisu literature naveli smo samo one reference koje su direktno utjecale na pojedine dijelove ovog udžbenika, možda više kao poticaj čitateljima da se i sami upuste u detaljnije istraživanje nekih tema koje smo mi samo vrlo kratko opisali.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. and Patel-Schneider, P.F. (ed.) (2007.) *The Description Logic Handbook: Theory, Implementation and Applications* (2nd Ed.), Cambridge University Press, 2007.

Bezinović, I. (2006.) Problem određenja znanja kao opravdanog istinitog vjerovanja, *Diskrepancija, studentski časopis za društveno-humanističke teme*, Sv.7, br.11. rujan 2006.

Braitenberg, V. (1984.) *Vehicles: Experiments in synthetic psychology*. Cambridge, MA: MIT Press, 1984.

Brooks, R. (2002.) *Flesh and Machines*, Pantheon Books, 2002.

Cullingord, R.E. (1978.) *Script application: Computer understanding of newspaper stories*, Ph.D.Thesis, Yale University, Comp.Sc.Dept. Research Report 150, 1979.

Dubois, D., Prade H. (1980.) *Fuzzy Sets and Systems: Theory and Applications,* ISBN 9780122227509, Elsevier,(1980. 1st ed.)

Gardner, H. (1983.) *Frames of Mind: The Theory of Multiple Intelligence,* ISBN 0-465-02510-2, Basic Books (1993 1st ed.)

Gene B., Durval C., Mills A., Data, Information, Knowledge and Wisdom, http://www.systems-thinking.org/dikw/dikw.htm

Gruber, T.R. (1993.) *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*, Int. Journal of Human-Computer Studies, Vol.43, No.5-6, Nov.1995, pp.907-928

Hayes-Roth F., Waterman D., Lenat, D. (1983.). *Building Expert Systems*. Addison-Wesley. 1983.

Haugeland, J. (1985.) *Artificial Intelligence: The Very Idea*, Cambridge, Massachusetts: MIT Press, 1985.

Hecht-Nielsen, R. (2005.) *Cogent Confabulation*, Neural Networks, vol. 18, no. 2, pp. 111--115, March 2005. [Online.] Available: http://dx.doi.org/10.1016/j.neunet.2004.11.003

Hecht-Nielsen, R. (2006.) *The Mechanism of Thought*, International Joint Conference on Neural Networks, Vancouver, Canada, pp. 419-426, July, 16--21 2006. http://dx.doi.org/10.1109/IJCNN.2006.246712

Khriyenko, O. (2019.) *TIES4520 Semantic Technologies for Developers (Course Curriculum)*, http://users.jyu.fi/~olkhriye/ties4520/

Kosko, B. (1993.) *Fuzzy Thinking: The New Science of Fuzzy Logic.* Hyperion. 1993.

Kowalski, R. (1979.) *Logic for Problem Solving*, Elsevier North Holland, 1979.

Luger, G.F. (2009.) *Artificial intelligence – Structures and Strategies for Complex Problem Solving*, Pearson Education, Inc. Boston, 2009 (6th Edition)

Mania, H., Guy, A. and Recht, B. (2018.) Simple random search provides a competitive approach to reinforcement learning, *NeurIPS - Advances in Neural Information Processing Systems, 31: Annual Conference on Neural Information Processing Systems 2018,* 3-8 December 2018, Montréal, Canada

Masterman, M. (1961.) Semantic message detection for machine translation, using Interlingua. *Proceedings of the 1961 International Conference on Machine Translation of Languages and Applied Language Analysis, Teddington, UK, Sept. 1961.*

Minsky, M. (1974.) *A Framework for Representing Knowledge,* MIT-AI Lab. Memo 306, June, 1974.

Moravec, H. (1988.) *Mind Children*, Harvard University Press, 1988.

Reiter, R. (2001.) *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, The MIT Press, 2001.

Russel, P. (1983.) *The Global Brain*, The Awakening Earth, U.K. 1983.

Russel, S., Norvig P. (2009.), *Artificial Intelligence: A Modern Approach (3rd Edition),* Pearson, 2009.

Rios L.H.O., Chaimowicz L. (2010.) A Survey and Classification of A* Based Best-First Heuristic Search Algorithms. In: da Rocha Costa A.C., Vicari R.M., Tonidandel F. (eds) *Advances in Artificial Intelligence – SBIA 2010.* SBIA 2010. Lecture Notes in Computer Science, vol 6404. Springer, Berlin, Heidelberg

Quillian, M.R. (1966.) *Semantic Memory*, PhD dissertation, Carnegie Institute of Technology (now CMU), skraćena verzija u Minsky,M. *Semantic Information Processing*, MIT Press (1968) pp. 227-270.

Quillian, M.R. (1967.) Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12(5):410-430, pretisak u Brachman and Levesque, *Readings in Knowledge Representation*. Los Altos, CA: Morgan Kaufmann, 1985.

Schank, R.C., Rieger, C.J. (1974.) Inference and the computer understanding of natural language, *Artificial Intelligence* 5(4):373–412.

Schank, R.C., Abelson, R. (1977.) *Scripts, Plans, Goals and Understanding*. Hillsdale, NJ: Erlbaum, 1977

Shadbolt, N. R., Smart, P. R. (2015.) Knowledge Elicitation. In J. R. Wilson & S. Sharples (Eds.), *Evaluation of Human Work (4th ed.).* CRC Press, Boca Raton, Florida, USA

Shapiro, C.S. (1992.) *Encyclopedia of Artificial Intelligence 2nd Edition*, John Wiley & Sons, Inc. New York, NY, USA, 1992.

Sternberg, R. J. (1985.) *Beyond IQ: A Triarchic Theory of Intelligence.* Cambridge: Cambridge University Press.

Stuard R., Peter N.(2003.) *Artificial Intelligence – a modern aproach*, Prentice Hall 2003.

Sun, X., Koenig, S. (2007.) The fringe-saving a* search algorithm - a feasibility study. In: *IJCAI*. (2007) 2391–2397

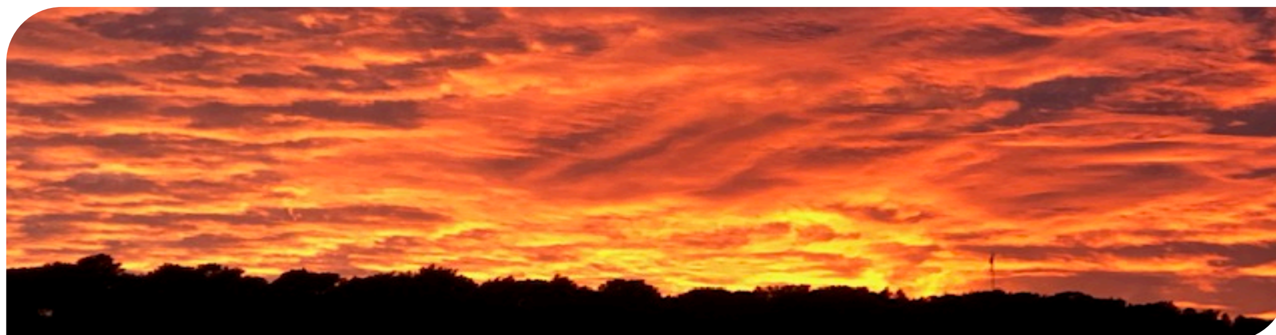
Sutton, R.S., Barto, A.G. (2017.) *Reinforcement Learning: An Introduction*, 2nd Ed., A A Bradford Book The MIT Press 2017.

Turner, R., (1984.) *Logics for Artificial Intelligence*, Ellis Horwood Limited, Chichester, England, 1984.

Watkins, C. J. C. H., Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Zadeh, L. A. (1965.) Fuzzy sets. *Information and Control*. **8** (3): 338–353.

DODATAK: PROGRAMIRANJE U PROGRAMSKIM JEZICIMA UMJETNE INTELIGENCIJE





Dodatak ovog udžbenika posvećen je programiranju u programskim jezicima umjetne inteligencije. Neki od njih, kao što su Lisp i Prolog, razvili su se u okvirima umjetne inteligencije, a bili su posebno značajni u razdobljima kada se umjetna inteligencija intenzivno razvijala. Danas je njihovo mjesto u najvećoj mjeri zauzeo Python, možda ne zato što je bolji, već zato što je univerzalni programski jezik koji se lako uči. U okviru programerske zajednice Pythona mogu se pronaći programska rješenja većine postupaka koje smo opisali u ovom udžbeniku. Dodatak je napisan u obliku vježbi koje se posljednjih godina izvode na FESB-u u okviru kolegija Umjetna inteligencija.

Najpoznatiji programski jezici umjetne inteligencije su Lisp, Prolog i Python, no programiranje metoda umjetne inteligencije nije ograničeno samo na njih. U praksi se metode umjetne inteligencije mogu programirati u bilo kojem programskom jeziku, no u nekima od njih će, zbog dostupne ili nedostupne podrške i dokumentacije, to biti jednostavnije ili složenije. Danas se za programiranje metoda umjetne inteligencije najčešće koristi Python jer je jednostavan, ima izvrsnu dokumentaciju, besplatan je, te je za njega dostupan velik broj biblioteka vezanih uz umjetnu inteligenciju.

Iako se Lisp i Prolog danas ne koriste toliko često kao prije, ipak ih je dobro naučiti jer zahtijevaju potpuno drugačiji pristup razmišljanju i programiranju od programskih jezika na koje je većina programera danas naviknuta, pa je savladavanje osnova ovih jezika sastavni dio svih uvodnih kolegija umjetne inteligencije.

U nastavku su osnove Lispa, Prologa i primjena Pythona u programima umjetne inteligencije. Kako su ovo ujedno i materijali laboratorijskih vježbi, organizirali smo ih u 10 vježbi. Sastavni dio vježbi su i zadaci za samostalno vježbanje.

VJEŽBA 1 - UVOD U PROGRAMIRANJE U LISP-U

Lisp (engl. *LISt Processor*) je jedan od najstarijih programskih jezika i jedan od najpopularnijih koji se tradicionalno koriste u umjetnoj inteligenciji. Osmišljen je 1958. g., i to samo dvije godine nakon što je utemeljeno područje umjetne inteligencije, a osmislio ga je ***John McCarthy***, istraživač koji je bio dio skupine znanstvenika koji su utemeljili područje umjetne inteligencije 1956. g.

Po kategoriji, Lisp spada u skupinu funkcionalnih programskih jezika (podržava komponiranje funkcija i rekurziju). Lisp je jezik opće namjene i drugi je po redu najstariji programski jezik visoke razine

(prvi je bio Fortran). Sjedinjuje svojstva strojnih jezika (potreba za poznavanjem detalja i sloboda manipuliranja podacima i programima) i viših jezika (jednostavnost programiranja). Lisp je dostupan u velikom broju tzv. dijalekata, tj. različitih verzija samoga sebe. Najpoznatiji dijalekti Lispa su *Common Lisp, Scheme i Emacs Lisp*. Postoji puno Lisp interpretera[108], a mi smo za primjere u tekstu ovog udžbenika već koristili *online* Lisp kompajler[109].

V1.1 Struktura Lispa

Lisp je programski jezik koji se temelji na listama. Svi izrazi (tj. forme) napisani u Lispu predstavljeni su u obliku listi, a liste su omeđene zagradama. Zbog mnogobrojnih zagrada u programskom kodu Lisp je zaradio status programskog jezika koji je složen za početnike jer se je lako *„izgubiti”* u svim tim zagradama. Važno je napomenuti da Lisp koristi ***prefiks notaciju*** (ime funkcije zadaje se prije argumenata). Na primjer, ako želimo obrađivati `listu (a b c ... n)`, tada vrijedi sljedeće pravilo: izraz *a* primjenjuje se na izraze `b c ... d` koji su njegovi argumenti. Na primjer, `(PLUS 2 3)` primjenjuje funkciju na argumente `2` i `3`. Iznimka se događa u slučaju kada ispred liste navedemo znak ', jer tada lista predstavlja podatak (npr. `'(1 2 3 4) i 'string`). Lisp nije osjetljiv na velika i mala slova.

Nakon upisane naredbe u Lisp interpreter, Lisp vraća rezultat izvršavanja te naredbe ili poruku o grešci.

Komentari u Lispu započinju znakom ;. Na primjer:

```
; ovo je komentar
```

Primjer ispisa znakovnog niza u Lispu:

```
(print "Pozdrav!")
```

Postavljanje vrijednosti varijabli u Lispu:

```
(setq a 6) ; a = 6
(setq b 'lubenica) ; b = "lubenica"
(setq c '(proljece ljeto jesen zima)) ; c = "proljece ljeto jesen zima"
```

Primijetite da se u gornjim primjerima jednostruki navodnik ' koristi za oznaku znakovnog niza. Dakle, ako se jednostruki navodnik nalazi ispred neke riječi, ta se riječ u Lispu koristi kao znakovni niz. Ako se jednostruki navodnik nalazi ispred zagrada, sve što se nalazi unutar tih zagrada koristi se kao znakovni niz.

Dva osnovna tipa podataka u Lispu su atomi i liste. ***Atomi*** su objekti koji se ne sastoje od drugih objekata i oni su za Lisp ono što su riječi u prirodnom jeziku. Oni s obje strane moraju biti ograđeni graničnicima (delimiterima), a to su praznina , točka zarez ;, točka . i zagrade (). Atomi mogu biti simboli ili brojevi.

Pomoću funkcije atom može se provjeriti je li neki objekt atom. Ova će funkcija vratiti T (istinito, engl. *True*) ako je objekt atom, te NIL ako nije. Na primjer:

```
> (atom 1)
T
> (atom '(1 2 3 4))
NIL
> (atom 'a)
T
```

[108] newLisp – http://www.newlisp.org, LispIDE – http://www.daansystems.com/lispide/, LispWorks – http://www.lispworks.com/

[109] https://rextester.com/l/common_lisp_online_compiler

Svaki atom ili lista može biti element liste. Bilo koji podskup liste također može biti lista. Atomi i liste zovu se još i ***s-izrazi (simbolički izrazi)***.

Brojevi u Lispu mogu biti cjelobrojni ili realni brojevi. Cjelobrojni brojevi započinju s + ili -. Realni brojevi imaju decimalnu točku i mogu biti napisani u eksponencijalnoj notaciji. Lisp podržava i kompleksne brojeve koji se pišu u obliku `#c(r i)`, gdje su `r` `i` `i` realni i imaginarni dijelovi kompleksnog broja. Primjeri brojeva u Lispu su: `23, -54, +5, 6.1234, 1.335e-16 i \#c(1.123e-10 0.12).`

Simboli u Lispu su nizovi znakova koji započinju sa slovom, a sastoje se od slova, brojeva i povlaka (npr. `i, a1, funkcija, funkcija-broj-jedan`). U Lispu se funkcija za pridruživanje vrijednosti simbolu naziva `setq`.

U Lispu su dostupne standardne aritmetičke funkcije: `+, -, * i /, ali i floor, ceiling, mod, sin, cos, tan, sqrt, exp, expt`, itd. Sve funkcije primaju za argument bilo koji tip broja, a vraćaju broj čiji tip ovisi o tipu argumenta. Jedino ograničenje za apsolutnu vrijednost cijelog broja je količina memorije u računalu. Potrebno je voditi računa o tome da su kalkulacije s velikim brojevima spore.

V1.2 Programiranje u Lispu

Lisp programi se interpretiraju. Kompajler je program koji uzima program napisan u skladu s pravilima nekog programskog jezika i pretvara ga u skup binarnih naredbi koje računalo može obraditi. Interpreter reagira izravno na naredbe i izvršava ih. Kada pokrenemo Lisp interpreter pojavljuje se prompt: '>'. Lisp interpreter čeka da korisnik unese dobro formirani Lisp izraz (formu) i odmah ga izvršava, ispisuje rezultat te se ponovo prikazuje prompt znak. Na primjer:

```
> (+ 1 2 1 2)
6
>
```

Kada interpreter pročita formu, izvršava tzv. ***čitaj-evaluiraj-ispiši*** (engl. *Read-Eval-Print)* ***petlju*** ili proceduru, koja se još naziva i ***temeljna petlja*** Lisp-a. `read` označava čitanje naredbe, `evaluate` označava izvršavanje naredbe, a `print` označava ispisivanje rezultata koji su se dobili nakon izvršavanja naredbe. Ako se dogodi da neka forma uzrokuje grešku, Lisp će vas postaviti u *debugger* mode. Većina *debuggera* odgovara na naredbu `help` ili `:help`. Za izlazak iz *debugging* moda dovoljno je utipkati `abort`.

Lisp omogućava dohvaćanje rezultata prethodnih naredbi na sljedeći način:

* dohvaća rezultat posljednje naredbe

** dohvaća rezultat prve od posljednje dvije naredbe

*** dohvaća rezultat prve od posljednje tri naredbe

Primjeri dohvaćanja rezultata prethodnih naredbi:

```
> (setq a 6) ; ovo korisnik upisuje
> 6 ; ovo je rezultat gornje naredbe koju Lisp ispisuje
> * ; ovo korisnik upisuje
> 6 ; ovo Lisp ispisuje
> (setq b 4)
> 4
> (setq c 10)
> 10
> **
```

```
> 4
> ***
> 4
```

Linija iza ; je komentar u Lispu i interpreter će je ignorirati. Lisp nije ***osjetljiv na velika i mala slova*** (engl. *Case Sensitive*).

V1.2.1 Setq

Lisp ima ugrađenu posebnu formu za pridruživanje vrijednosti simbolu koja se zove `setq`. Na primjer, u C-u bi se pisalo x = 5 kada bismo željeli varijablu x postaviti na 5, a u Lispu bi se pisalo `(setq x 5)`. Primjeri korištenja naredbe setq u Lispu:

```
> (setq a 5) ; pohrana broja 5 u simbol a
> a ; dohvaćanje vrijednosti simbola
5
> (setq a 'hello) ; pohrana simbola
HELLO
> (setq lista1 '(1 2 3 4)) ; inicijalizacija liste
(1 2 3 4)
> b ; pokušaj korištenja nepostojećeg simbola
Error: Attempt to take the value of the unbound symbol B
```

V1.2.2 Cons

`cons` je zapis s dva polja. Polja se zovu `car` (engl. *Contents of Address Register*) i `cdr` (engl. *Contents of Decrement Register)* iz povijesnih razloga, te odgovaraju prvom polju i ostatku (engl. *first* i *rest*). Ova su polja shematski prikazana na slici V1-1.



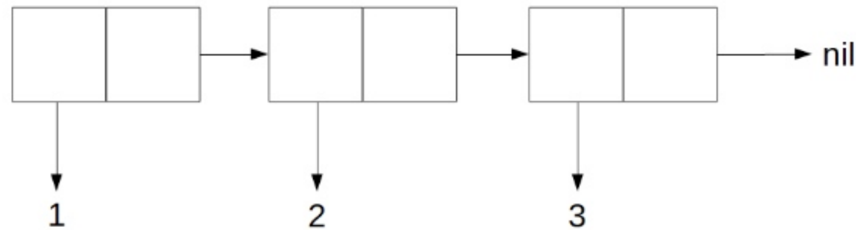
Slika V1-1. Polja car i cdr

Primjeri korištenja naredbe `cons` u Lispu:

```
> (cons 4 5) ; alocira se cons polje, car se postavlja na 4 a cdr na 5
(4 . 5)
> (cons (cons 4 5) 6) ; car se sastoji od cons, koji se sastoji od car i cdr
((4 . 5) . 6)
> (car (cons 4 5)) ; dohvati car
4
> (cdr (cons 4 5)) ; dohvati cdr
5
```

V1.2.3 Liste

Od `cons` se mogu graditi razne strukture, kao na primjer liste ili stabla. Vezana lista može se izgraditi na način da `car` jednog `consa` pokazuje na jedan element liste, a `cdr` pokazuje ili na drugi element ili na `nil`. Takva vezana lista u Lispu je shematski prikazana na slici V1-2.



Slika V1-2. Vezana lista u Lispu

Vezane liste u Lispu mogu se kreirati na jednostavniji način pomoću funkcije `list`. Na primjer:

```
> (list 4 5 6)
(4 5 6)
```

Funkcija `listp` vraća `T` (istinito, engl. *True*) ako je argument funkcije lista, te `NIL` ako nije. Na primjer:

```
> (setq a '(1 2 3 4 5))
(1 2 3 4 5)
> (listp a)
T
> (listp 'a)
NIL
```

Ako ispred liste nije `quote (')` onda se prvi element liste tretira kao ime funkcije.

Lisp ispisuje liste na poseban način (ne ispisuje neke točke i zagrade), po sljedećim pravilima:

- ako je `cdr nil` ne ispisuju se točke
- ako je `cdr cons-a A cons B`, ne ispisuju se točke za `A` ni zagrade za `B`.

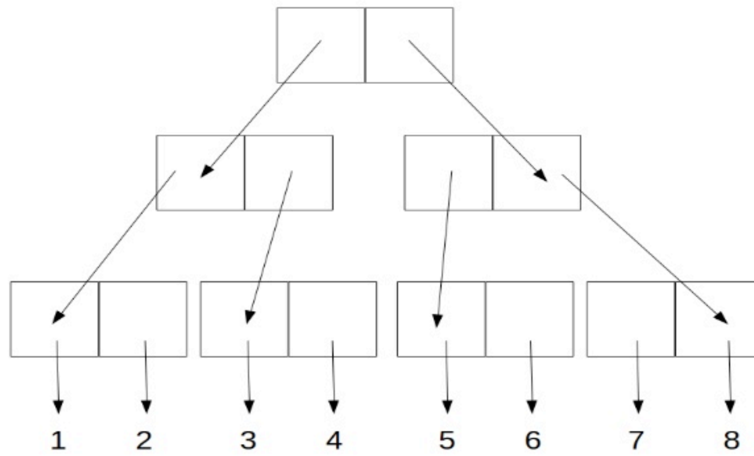
Primjeri gore navedenih pravila:

```
> (cons 4 nil)
(4)
> (cons 4 (cons 5 6))
(4 5 . 6)
Car i cdr nil liste su nil. Nil označava listu bez elementa. Na primjer:
> (cons 4 (cons 5 (cons 6 nil)))
(4 5 6)
```

Gornja naredba je ekvivalentna naredbi `(list 4 5 6)`.

V1.3 Zadaci

Zadatak 1.3.1. Napišite izraz u Lispu čiji će ispis odgovarati slici V1-3, te odredite što su `car` i `cdr` te strukture.



Slika V1-3. Slika vezana uz zadatak 1

Zadatak 1.3.2. Izdvojite broj 5 iz strukture iz prvog zadatka.

Zadatak 1.3.3. Koristeći `car` i `cdr` izvucite riječ *svemir* iz sljedećih konstrukcija:

```
(svemir je pun zvijezda)
(svemir (((je) beskonacan)))
((((svemir) je) inspirirao) brojne) poznate znanstvenike)
(mate (i ivana) vole promatrati svemir)
```

Zadatak 1.3.4. Koristeći `cons` napraviti sljedeću strukturu:

`(ovo (je jedna) (struktura (sa zagradama))).`

VJEŽBA 2 - NAPREDNO PROGRAMIRANJE U LISPU

V2.1 Napredan rad s listama

Funkcijom `list` u Lispu se kreira lista. Na primjer:

```
> (list a b c d)
> (list 'a 'b 'c 'd)
```

Lista se može ponašati i kao stog, pa se mogu koristiti funkcije `push` i `pop`. Na primjer:

```
> (setq a nil)
NIL
> (push 4 a)
(4)
> (push 5 a)
(5 4)
> (pop a)
5
> a
(4)
> (pop a)
4
```

```
> (pop a)
NIL
> a
NIL
```

Primjeri funkcije `atom`:

```
> (setq a '(l i s t a))
(L I S T A)
> (atom a)
NIL
> (atom 'l)
T
```

Primjeri funkcije `listp`:

```
> (listp a)
T
> (listp 'a)
NIL
Primjeri funkcije length:
> (length a)
5
> (length 'a)
*** - LENGTH: A is not a sequence
Primjeri funkcije append:
> (append a a)
(L I S T A L I S T A)
> (length (append a a))
10
> (length '(append a a))
3
```

Primjeri:

```
> (cons 'a '(b c d))
(a b c d)
> (list 2 3 4)
(2 3 4)
> (cons 1 *)
(1 2 3 4)
> (list 'a '(a s d f))
(a (a s d f))
> (setq a *)
(A (A S D F))
> (cons 'a a)
(A A (A S D F))
```

V2.2 Booleovi izrazi

Lisp koristi simbol `nil` za označavanje neistine. Sve drugo osim nil znači istinu, osim ako imamo poseban razlog za korištenje simbola T za označavanje istine.

Korisno:

- `nil` - prazna lista ili logička neistina
- `T` - eksplicitna oznaka logičke istine
- `null` - funkcija koja provjerava je li argument `nil`
- `not` - negacija, ako je argument `nil` vraća `T`, u protivnom vraća `nil`.

Primjeri:

```
> (null nil)
T
> (not nil)
T
> (null ())
T
> (not ())
T
> (null '(a s))
NIL
> (not '(a s))
NIL
```

Lisp ima standardne logičke funkcije (`and`, `or` i `not`). `and` i `or` djeluju kao kratki spoj. `and` izvršava sve argumente dok se ne pojavi jedan koji se izvršava u `nil`, a `or` izvršava sve argumente dok se ne pojavi jedan koji se izvršava u `T`.

Primjer:

```
> (and 1 2 3 4)
4
> (and 1 (cons 'a '(b)) (rest '(a)) (setf y 'hello))
NIL
> (or nil nil 2 (setf y 'goodbye))
2
> (or (rest '(a)) (equal 3 4))
NIL
```

V2.3 Uvjetni izrazi

Lisp ima ugrađene predikate `<`, `>`, `<=` i `>=`. Brojčana jednakost označava se s `=`. Funkcija `equal` daje istinu za dvije `cons` strukture ako i samo ako su njihovi `car` dijelovi jednaki i njihovi `cdr` dijelovi jednaki. Daje istinu za dvije strukture ako i samo ako su one istog tipa i ako su njihova odgovarajuća polja jednaka.

Pravila:

- `(= x y)` - istina ako su x i y numerički jednaki
- `(equal x y)` - istina ako su ispisi vrijednosti x i y varijabli jednaki, tj. imaju jednaku vrijednost i strukturalno su jednaki
- `(eq x y)` - istina ako su x i y isti objekt u memoriji
- `(eql x y)` - koristi se kao eq ako su x i y simboli, ili kao = ako su x i y brojevi.

Općenito, `funkcije = i equal` se koriste češće nego `eq` i `eql`.

Primjeri:

```
> (setq a '(1 2 3))
(1 2 3)
> (eq a '(1 2 3))
NIL
> (setq b a)
(1 2 3)
> (eq a b)
T
> (= 3 3.0)
T
> (= 3/1 6/2)
T
> (eq 3 3.0)
NIL
> (eq 3 6/2)
T
> (eq 3.0 6/2)
NIL
> (eql 3.0 3/1)
NIL
> (eql 3 6/2)
T
> (equal 3 3)
T
> (equal 3 3.0)
T
```

Lisp ima nekoliko posebnih formi za uvjetno izvršavanje. Najjednostavnija je `if` forma. Prvi argument uzrokuje hoće li se izvršiti drugi ili treći argument. If forma ima sljedeći oblik:

`(if (uvjet) (then dio) (else dio))`

Primjeri:

```
> (if t 5 6)
5
> (if nil 5 6)
```

```
6
> (if 4 5 6)
5
> (setq a 7)
7
> (setq b 0)
0
> (setq c 5)
5
   > (if (> a 5)(progn (setq a (+ b 7)) (setq b (+ c 8))) (setq b 4) )
13
```

Ako je potrebno staviti više od jednog izraza u `then` ili `else` dio može se koristiti `progn` posebna forma. Progn izvršava svaku naredbu u svom tijelu i vraća vrijednost posljednje.

If naredba kojoj nedostaje ili `then` ili `else` dio može se napisati koristeći `when` ili `unless` posebne forme, koje imaju sljedeće oblike:

```
(when (uvjet) (then dio))
(unless (uvjet) (else dio))
```

Primjeri:

```
> (when t 3)
3
> (when nil 3)
NIL
> (unless t 3)
NIL
> (unless nil 3)
3
```

Za razliku od `if` naredbe `when` i `unless` naredbe dopuštaju veći broj naredbi u svom tijelu. Na primjer, `(when x a b c)` je identičan s `(if x (progn a b c))`.

Za složenije uvjetne izraze koristimo `cond` posebnu formu koje je ekvivalentna s `if ... else if ... else` konstrukcijom. Sintaksa: nakon ključne riječi `cond` piše se niz cond izraza. Cond izraz je lista koje se sastoji od uvjeta na prvom mjestu i liste akcija koja se izvršavaju ako je uvjet istinit. Izvršavaju se samo akcije iz prvog cond izraza koji se razvije u istinu, sve ostale se preskaču.

```
(cond(uvjet1 akcije...)(uvjet2 akcije2 ...)...(t else-dio ...) )
```

Primjeri:

```
> (setq a 3)
3
> (cond
((evenp a) a) ; ako je a paran - vratiti a
((> a 7) (/ a 2)) ; ako je a neparan i veći od 7 - vratiti $\frac{a}{2}$
((< a 5) (- a 1)) ; ako je a manji od 5 - vratiti $a-1$
(t 17) ; ništa od ponuđenog - vratiti 17
```

```
)
2
```

Ako u cond izrazu nedostaje akcija, cond vraća ono u što se razvije uvjet. Na primjer:

```
> (cond ((+ 3 4)))
7
```

Case naredba u Lispu je kao switch naredba u C-u, te ima sljedeći oblik:

(case simbol((lista vrijednosti1) akcija 1)((lista vrijednost2) akcija 2) ... (otherwise default akcija))

Primjer:

```
> (setq x 'b)
B
> (case x
(a 5)
((d e) 7)
((b f) 3)
(otherwise 9)
)
3
```

otherwise izraz na kraju znači da x nije neka od ponuđenih vrijednosti.

V2.4 Funkcije

Kod definiranja funkcija u Lispu vrijedi sljedeće pravilo:

(DEFUN ime funkcije(lista argumenata)(implementacija funkcije))

Primjer definiranja funkcije:

```
> (defun bar(x) (setq x (* x 3)) (setq x (/ x 2)) (+ x 4) )
BAR
> (bar 6)
13
```

Ako funkcija ima više izraza unutar sebe, onda će vratiti rezultat posljednjeg izraza.

Primjeri funkcija:

```
> (+ 3 4 5 6) ;ova funkcija uzima bilo koji broj argumenata
18
> (+ (+ 3 4) (+ (+ 4 5) 6))
22
> (defun foo(x y)(+ x y 5)) ; definiranje funkcije
FOO
> (foo 5 0) ; poziv funkcije
10
```

Kada pozivamo funkciju moramo joj dati onoliko argumenata koliko smo specificirali prilikom definicije funkcije. Na primjer:

```
> (defun moj-zbroj (x y) (+ x y))
MOJ-ZBROJ
> (moj-zbroj 2 3)
5
> (moj-zbroj 2)
error: too few arguments
```

U Lispu je moguće specificirati i neobavezne argumente funkcije. Svaki argument nakon simbola `&optional` je neobavezan. Neobaveznim argumentima mogu se pridijeliti predefinirane vrijednosti (engl. *Default)*. Primjeri:

```
> (defun bar (x &optional y) (if y x 0))
BAR
> (defun baaz (&optional (x 3) (z 10)) (+ x z))
BAAZ
> (bar 5)
0
> (bar 5 t)
5
> (baaz 5)
15
> (baaz 5 6)
11
> (baaz)
13
```

Funkcija može primati neodređeni broj argumenata. Ako listu argumenata završimo s parametrom `&rest`, Lisp će pokupiti sve argumente koji nisu već vezani u listu i vezati parametar `&rest` za listu.

Primjeri:

```
> (defun foo (x &rest y) y)
FOO
> (foo 3)
NIL
> (foo 4 5 6)
(5 6)
```

Kada se poziva funkcija argumenti se mogu davati bilo kojim redom jer su označeni ključnom riječi `&key`.

Primjeri:

```
> (defun foo (&key x y) (cons x y))
FOO
> (foo:x 5:y 3)
(5 . 3)
> (foo:y 3:x 5)
(5 . 3)
```

```
> (foo:y 3)
(NIL . 3)
> (foo)
(NIL)
```

Parametar `&key` može imati default vrijednost:

```
> (defun foo (&key (x 5)) x)
FOO
> (foo:x 7)
7
> (foo)
5
```

V2.5 Lokalne i globalne varijable

Primjeri:

```
> (defun y-plus (x) (+ x y))
Y-PLUS
> (setq y 2)
2
> (y-plus 4)
6
> (setq x 5)
5
```

```
> (y-plus 23)
25
> x
5
> (setq y 27)
27
> (y-plus 43)
70
```

V2.6 Naredbe ponavljanja (iteracije)

Najjednostavnija naredba ponavljanja u Lispu je `loop`. `Loop` izraz izvršava se iznova i iznova dok ne naiđe na posebnu formu `return`. `Loop` mora doći u kombinaciji s `return` inače će se razviti u beskonačnu petlju. Forma `return` vraća vrijednost izraza unutar `return` forme ili `nil`, te ima sljedeći oblik: `(return izraz)`.

Primjer 1:

```
> (setq a 4)
4
> (loop (setq a (+ a 1)) (when (> a 7) (return a)) )
```

```
8
```

Primjer 2:

```
> (loop (setq a (- a 1)) (when (< a 3) (return)) )
NIL
```

Forma `dolist` redom veže varijablu za elemente liste i završava izvršavanje kada dođe do kraja liste. Ima sljedeći oblik:

```
(dolist (simbol lista) akcije koje se izvršavaju za svaki član liste)
```

Primjer:

```
> (dolist (x '(a b c)) (print x))
A
B
C
NIL
```

Forma `dolist` uvijek vraća `nil`. U gornjem primjeru X nikada nije bio nil, no nil je ipak vrijednost koju je gornji izraz vratio nakon izvršavanja.

Forma `do` je najsloženija naredba ponavljanja. Do izraz izgleda ovako:

```
(do (
( varijabla1 poč-vrij-var1 (izraz-za-promjenu-var1) )
( varijabla2 poč-vrij-var2 (izraz-za-promjenu-var2) )
...)
((uvjet-prekida petlje) (povratna-vrijednost))
...
Tijelo-petlje
... )
```

Prvi dio do forme specificira koje varijable vezati, njihove početne uvjete, te kako ih mijenjati. Drugi dio do forme specificira uvjet prekida i konačnu vrijednost. Posljednji dio do forme je tijelo izraza. Do forma veže varijable na početne vrijednosti, kao i forma `let`, te tada provjerava uvjet prekida. Kad se uvjet prekida razvija se u `nil`, te se iznova izvršava tijelo, a kada uvjet postane istina, vraća se vrijednost forme povratne vrijednosti.

Primjer:

```
> (do ((x 1 (+ x 1)) (y 1 (* y 2))) ((> x 5) y) (print y) (print 'working) )
1
WORKING
2
WORKING
4
WORKING
8
WORKING
16
WORKING
```

```
32
```

V2.7 Rekurzivna funkcija

Primjer 1:

```
> (defun factorial (n) (if (<= n 1) 1 (* n (factorial (- n 1)))) )
FACTORIAL
> (factorial 5)
120
> (defun a (x) (if (= x 0) t (b (- x)))) ; mutually recursive functions
A
> (defun b (x) (if (> x 0) (a (- x 1)) (a (+ x 1))))
B
> (a 5)
T
```

Primjer 2:

```
> (defun listanaopako (l1) (setq a nil) (dolist (x l1) (if (atom x) (push x a) (push
(listanaopako x) a) ) ) a)


(trace listanaopako)
(listanaopako '( 1 2 3 (3 4) 5 ))
1. Trace: (LISTANAOPAKO '(1 2 3 (3 4) 5))
2. Trace: (LISTANAOPAKO '(3 4))
2. Trace: LISTANAOPAKO $\implies$ (4 3)
1. Trace: LISTANAOPAKO $\implies$ (5 (4 3) 4 3)
(5 (4 3) 4 3) ...
```

V2.8 Ispis

Neke funkcije zahtijevaju izlaz, a za to je najjednostavnije korištenje forme **print** koja ispisuje argument i vraća argument.

```
> (print 3)
3
3
```

U gornjem primjeru prvi 3 je ispisan, a drugi je vraćen.

Za složenije ispise koristi se forma **format**:

```
> (format t "An atom: ~S~\%and a list: ~S~\%and an integer: ~D~\%" nil (list 5) 6)
An atom: NIL
and a list: (5)
and an integer: 6
```

V2.9 Zadaci

Zadatak 2.9.1. Definirati funkciju koja računa hipotenuzu pravokutnog trokuta.

Zadatak 2.9.2. Napisati funkciju koja koristeći `car` i `cdr` vraća četvrti element liste.

Zadatak 2.9.3. Definirati funkciju koja prima 1, 2 ili više argumenata:

ako ima 1 argument vraća taj argument pomnožen s 1,

ako ima 2 argumenta vraća umnožak ta 2 broja,

ako ima više argumenata vraća listu kojoj je na prvom mjestu umnožak prva 2 argumenta a ostatak liste su preostali argumenti.

Zadatak 2.9.4. Definirati funkciju koja prima 1 ili više argumenata:

ako ima 1 argument vraća taj argument,

ako ima više argumenata vraća listu kojoj je na prvom mjestu umnožak prva 2 argumenta a ostatak liste su preostali argumenti.

Zadatak 2.9.5. Napisati funkciju koja koristeći do petlju vraća listu prvih 15 kubnih vrijednosti `(1 8 27 64 ... 3375)`.

Zadatak 2.9.6. Napisati funkciju **moj-kalkulator** koja prima dva broja i opcionalno operaciju `(* / \%)`. Funkcija vraća rezultat izvršavanja operacije na dva broja. Ako korisnik ne navede operaciju koristi se zbrajanje.

Primjeri izvršavanja funkcije:

```
> (moj-kalkulator 2 3 '*)
6
> (moj-kalkulator 2 3)
5
> (moj-kalkulator 10 50 \%) ; 10 posto od 50
5
```

Zadatak 2.9.7. Definirati funkciju `add12` koja svakom članu liste dodaje broj 12.

Primjer izvršavanja funkcije:

```
> (add12 '(1 2 3 4))
(13 14 15 16)
```

Zadatak 2.9.8. Napisati rekurzivnu funkciju `postoji` koja vraća `T` ako se dati atom pojavljuje bilo gdje u nekom izrazu, a `NIL` inače.

Primjer provjere postoji li varijabla x u nekom izrazu:

```
> (postoji 'x '(sqrt (/ (+ (exp x 2) (exp y 2)) 2)))
T
```

VJEŽBA 3 - PROLOG

Predikatna logika je matematički formalni jezik koji koristimo za predstavljanje znanja, te predstavlja logički sustav u kojemu se mogu izraziti i matematički izrazi i rečenice iz prirodnog jezika. Predikatna logika sadrži i pravila zaključivanja pomoću kojih se izvode nove rečenice iz skupa postojećih. Korištenje formalnog jezika je bitno zbog njegove nedvosmislenosti i zbog lakšeg zapisivanja pravila. Prolog je programski jezik kojim je moguće opisati i riješiti određen problem na način opisan predikatnom logikom.

Programski jezik ***Prolog***[110] (*fran. PROgrammation en LOGique, engl. PROgramming in LOGic*) osmislio je ***Alain Colmerauer*** (1941.-2017.) 1973. g. Programi napisani u Prologu sastoje se od liste činjenica i pravila (tj. od baze znanja), te od upita vezanih uz te informacije. Zbog ovih činjenica Prolog se najčešće koristi za izgradnju stručnih (ekspertnih) sustava.

Pisanje Prolog programa ssastoji se od:

- deklariranja tvrdnji (činjenica) o objektima i njihovim međusobnim odnosima
- deklariranja pravila o objektima i njihovim međusobnim odnosima i
- postavljanja upita tj. ciljeva o objektima i njihovim međusobnim odnosima.

Prolog program je lista činjenica i pravila, i to je sve što program sadrži. Zatim postavljamo upite. Prolog sadrži ***stroj za zaključivanje*** (engl. *Inference Engine)* koji je procedura koja prolazi kroz listu pravila i činjenica i pokušava odgovoriti na upit koristeći dedukcijsku logiku.

Za vježbanje Prologa najbolje je koristiti on – line verziju ***SWI-Prologa*** (https://swish.swi-prolog.org/) koji ima izvrsnu podršku. S njim smo se već susretali u primjerima koje smo u okviru ovog udžbenika davali u Prologu.

Programi u Prologu mogu se pisati u Python *shell-u* (koji započinje znakovima ?-) ili u datotekama s ekstenzijom ***.pl***. Ako se programi pišu u datoteci s ekstenzijom .pl, onda se ta datoteka u Python *shell* učitava sa ***consult(imeDatoteke)*** ili ***[imeDatoteke]***.

Prolog podržava dva načina unosa komentara. U prvome načinu komentari započinju znakom %, a u drugom načinu komentari su omeđeni znakovima /* i */ (tj. jednaki su komentarima u programskom jeziku C). Drugi način zapisa komentara koristi se u slučajevima kada je potrebno brzo komentirati više linija koda.

Primjeri komentara u Prologu:

```
% ovo je prvi komentar
/* ovo je drugi komentar */
/* ovo je početak trećeg komentara
* ovo je sredina trećeg komentara
* ovo je kraj trećeg komentara */
```

Primjer ispisa znakovnog niza u Prologu:

```
writeln('Pozdrav!').
```

V3.1 Predikati

Predikati se koriste za opisivanje pravila, činjenica i upita. Primjeri:

```
Cold.
likes(dora,dancing).
difficult(physics).
5 < 12.
```

Svi predikati imaju sljedeći oblik:

```
functor(arg1, arg2, arg3, ...).
```

gdje je:

[110] Prolog – Computer Language. URL: https://www.britannica.com/technology/PROLOG

- `functor` - ime predikata koje sami zadajemo - može biti relacija ili predikat,
- `argX` - argumenti mogu biti:
    - konstante - brojevi ili atomi (obično započinju malim slovom, za razliku od varijabli)
    - varijable - započinju velikim slovom
    - strukture
    - liste - posebni oblik strukture.

Napomena: između `functora` i lijeve zagrade ne smije doći razmak.

Sami pridajemo značenje predikatu i moramo ga koristiti konzistentno - uvijek s istim brojem i značenjem argumenata. Na primjer:

```
likes(nina,jabuke)
```

Gornji primjer znači da Nina voli jabuke. Functor '`like`' ima 2 argumenta od kojih je prvi volitelj a drugi voljenik. Da bismo održali jednoznačnost pomažu nam komentari. Trebali bismo datoteci koja sadrži pravila dodati:

```
/* likes(liker,likee). */
```

Ako neku činjenicu baza podataka ne sadrži automatski se smatra neistinom. Postoji i eksplicitan način za izražavanje neistine:

```
likes(nina,jabuke):- fail,!.
```

Predikati se koriste za izražavanje nekoliko općih jezičnih smislenosti:

- odnos između dvaju entiteta - prijedlog, imenica ili glagol
- opis entiteta - pridjev, često ima samo jedan argument
- označavanje radnje - glagol
- predikati bez argumenta koriste se za označavanje općenitog stanja „svijeta".

Primjeri:

```
/* odnos između dva entiteta */
boss(ivana,matea). /* Ivana je šefica Matei. */
on(olovka,stol). /* Olovka je na stolu. */
rules(elizabeth,engleska). /* Elizabeth vlada Engleskom. */

/* opis entiteta */
alkoholan(viski). /* Viski po sastavu sadrži alkohol. */
tezak(ui). /* Umjetna inteligencija je težak kolegij. */

/* označavanje radnje */
eat(dječak,sladoled). /* Dječak jede sladoled. */

/* predikati bez argumenta */
game\_running. /* Igra traje, nije zaustavljena. */
```

V3.2 Rad sa Prologom

Nakon pokretanja Prologa pojavi se:

```
?-
```

Učitavanje baze:

```
?- consult(prvi) /* 1. način */
?- [prvi] /* 2. način */
```

Ispis svih činjenica iz učitane baze:

```
?- listing
```

Izlazak iz programa:

```
?- halt
```

V3.2.1 Upiti

Upiti se u Prologu koriste za pretraživanje baze znanja. Činjenica koja nam je nepoznata i na koju želimo saznati odgovor u Prologu se označava sa velikim slovom. Činjenicu koja nam je nevažna označavamo znakom _.

Primjer baze znanja:

```
roditelj(marko, šime). % Marko je roditelj Šime.
roditelj(ana, petra). % Ana je roditelj Petre.
sestra(marija, anđela). % Marija je sestra Anđele.
brat(stipe, matija). % Stipe je brat Matije.
brat(stipe, dominik). % Stipe je brat Dominika.
roditelji(branimir, mia, ivana). % Branimir i Mia su roditelji Ivane.
roditelji(domagoj, mia, mirta). % Domagoj i Mia su roditelji Mirte.
```

Primjeri upita:

```
?- roditelj(X, šime). % X označava činjenicu koju želimo saznati.
X = marko % Odgovor koji dobijemo u Prologu.
?- brat(stipe,X).
X = matija
X = dominik
?- roditelj(X, petra).
X = ana
?- roditelji(X,Y,ivana).
X = branimir,
Y = mia
?- roditelji(_, mia, X). % Tražimo djecu Mije bez obzira na oca.
X = ivana
X = mirta
?- brat(stipe, matija). % Pitamo Prolog je li ova činjenica istinita.
true % Prolog odgovara pozitivno.
?- brat(stipe, luka). % Ova činjenica ne postoji u bazi znanja.
```

```
false % Prolog odgovara negativno.
```

V3.2.2 Pravila – relacije između predikata

Pravila izražavaju povezanost predikata. Na primjer:

`like(X,Y):- love(X,Y).`

Gornji primjer znači da ako nešto volimo onda nam se to sviđa. Odnosno:

if love(X,Y) then like (X,Y).

Oznaka `':-'` znači *'if'*, ali imamo obrnutu notaciju.

Ponekad nam je potrebno više uvjeta. Ako uvjete razdvojimo s `','` zarez se ponaša kao ***logičko 'i'***. Na primjer:

`zaključak:- uvjet1,uvjet2,uvjet3.`

U gornjem primjeru Zaključak je istinit ako su istiniti uvjeti `uvjet1`, `uvjet2` i `uvjet3`. Primjer:

`can_enter(X):- high_school_grad(X),gpa >=2.0,has_money(X).`

Zaključak i uvjet su predikati. Uvjeta može biti 0 ili više.

Logičko 'ili' se formira na način da više pravila ima iste posljedice. Na primjer:

`roditelj(X,Y):-otac(X,Y).`

`roditelj(X,Y):-majka(X,Y).`

Također možemo definirati i ***iznimke***:

`ptica(X):- leti(X), ima(X,perje).`

Ali postoje ptice (`pingvin i noj`) koje ne lete, pa moramo uključiti i iznimke:

`ptica(pingvin).`

`ptica(noj).`

Pravila u Prologu dopuštaju i ***rekurziju***. Dopušteno je imati isti predikat s lijeve i desne strane implikacije. Na primjer:

`predak(X,Y):- roditelj(X,Z), predak(Z,Y).`

Gornji primjer kaže da ako je Y predak od Z i ako je Z roditelj od X, onda je Y predak od X. Pogledajmo sada jedan jednostavni primjer definiranja pravila i zaključivanja s njima:

```
otac(marko, julija). % Marko je otac Julije.
majka(viktorija, julija). % Viktorija je majka Julije.
roditelj(X,Y):- otac(X,Y). % Ako je X otac od Y, onda je X roditelj od Y.
roditelj(X,Y):- majka(X,Y). % Ako je X majka od Y, onda je X roditelj od Y.
?- roditelj(X,julija). % Primjer upita.
X = marko % Prologov odgovor.
X = viktorija % Prologov odgovor.

djed(filip,marija). % Filip je djed Marije.
baka(lucija,marija). % Lucija je baka Marije.
majka(marijeta,marija). % Marijeta je majka Marije.
otac(luka, marija). % Luka je otac Marije.
```

```
predak(X,Y):- djed(X,Y). % Ako je X djed od Y, onda je X predak od Y.
predak(X,Y):- baka(X,Y). % Ako je X baka od Y, onda je X predak od Y.
predak(X,Y):- majka(X,Y). % Ako je X majka od Y, onda je X predak od Y.
predak(X,Y):- otac(X,Y). % Ako je X otac od Y, onda je X predak od Y.
?- predak(X,marija). % Primjer upita.
X = filip % Prologov odgovor.
X = lucija % Prologov odgovor.
X = marijeta % Prologov odgovor.
X = luka % Prologov odgovor.

djed(ivan, mia).
baka(jelena, mia).
% Ako je Y baka od X, i ako je Z djed od X, onda je X unuka od Y i Z.
unuka(X, Y, Z):- baka(Y, X), djed(Z, X).
?- unuka(mia, jelena, ivan). % Je li Mija unuka Jelene i Ivana?
true % Prolog odgovara pozitivno.
?- unuka(mia, X, Y). % Tko su baka i djed Mije?
X = jelena,
Y = ivan
```

V3.3 Logičke operacije u Prologu

Pravila u Prologu su implementacija logičkog operatora implikacije:

```
\begin{equation}
A \implies B \iff (\neg A) \lor B
\end{equation}
```

U tablici V-1 dana je tablica istine za logičke operacije, među kojima i za implikaciju.

Tablica V-1. Tablica istine za logičke operacije

X	***Y***	***ILI***	***I***	***IMPLIKACIJA***	***NEGACIJA***
T	*T*	*T*	*T*	*T*	*F*
T	*F*	*T*	*F*	*F*	*F*
F	*T*	*T*	*F*	*T*	*T*
F	*F*	*F*	*F*	*T*	*T*

Primjeri implikacije:

- *Ako konji imaju krila tada slonovi lete.*
- *Ako pada kiša vrata fakulteta su otvorena.*
- *Ako je 3 paran broj tada je 3 djeljiv s 2.*

- *Ako smo živi tada dišemo.*

Pravila u Prologu su implicitno univerzalno kvantificirana. Sve varijable koje se pojavljuju u drugim predikatima osim u predikatu lijeve strane su egzistencijalno kvantificirane. Na primjer:

```
ptica(X):-imaperje(X).
```

Gornji primjer kaže da za svaki `X`, ako je istina da `X` ima perje, onda je istina da je `X` ptica. Sve što ima perje mora biti ptica. Nadalje, lijeva strana gornjeg primjera označava se s ***LHS*** (engl. *Left Hand Side*), a desna s ***RHS*** (engl. *Right Hand Side*).

Pravila u Prologu moraju biti općenita (moraju uvijek vrijediti) te zbog toga varijable moraju biti univerzalno kvantificirane.

Poredak implikacije je *RHS* → *LHS*, što znači da desna strana implicira lijevu stranu. To ne znači da vrijedi i obrnuto, tj. ne vrijedi *LHS* → *RHS*. Na primjer, *'(pada kiša) → (ne sja sunce)'* je uglavnom istinito, ali *'(ne sija sunce) → (pada kiša)'* nije.

U Prologu postoje i pravila koja imaju više izraza na RHS, npr. *'Moji prijatelji su tvoji prijatelji'*, tj. ako smo ja i ti prijatelji svi tvoji prijatelji su i moji prijatelji. U Prologu bi ovo zapisali kao:

```
prijatelj(ja,netko):- prijatelj(ja,ti), prijatelj(ti,netko).
```

U gornjem primjeru varijable `'ja'` i `'netko'` su **univerzalno kvantificirane**. To znači da za sve `'ja'` i za sve `'netko'` vrijedi sljedeće: ako smo `'ja'` i `'ti'` prijatelji i `'ti'` i `'netko'` ste prijatelji onda smo `'ja'` i `'netko'` prijatelji. Varijabla `'ti'` je **egzistencijalno kvantificirana** što znači sljedeće: ako postoji `'ti'` koji je prijatelj od `'ja'`, a ako je `'ti'` prijatelj s nekim `'netko'`, tada su `'ja'` i `'netko'` također prijatelji. Ovo pravilo vrijedi bez obzira tko je `'ja'` i tko je `'netko'`, dok god postoji neki `'ti'` na koji se ovo može primijeniti.

V3.4 Trace mehanizam

Prolog ima ugrađen mehanizam praćenja pretraživanja baze koji se pokreće na sljedeći način:

```
?-trace
```

Postavljanjem upita ispisuju se svi koraci prilikom pretraživanja baze. Trace mehanizam se zaustavlja na sljedeći način:

```
?-notrace
```

V3.5 Primjer složenijeg zaključivanja u Prologu

Pogledajmo jedan složeniji primjer zaključivanja u Prologu. Radi se o ***Einsteinovoj zagonetki*** (engl. *Einstein' Riddle)* koju smo već spominjali u poglavlju 3. Legenda kaže da ju je osmislio ***Albert Einstein*** (1879.-1955.) kao dijete, ali to nikada nije dokazano. Varijacijua ove zagonetke poznata kao ***zebra zagonetka*** (engl. *Zebra Puzzle)* pripisuje se ***Lewisu Carrollu*** (1832.-1898.), engleskom piscu i matematičaru, autoru *„Alise u zemlji čudesa"*, ali ni to nije potpuno potvrđeno. Ovdje ćemo prikazati malo izmijenjenu varijantu zagonetke koju smo prilagodili današnjem vremenu u kojem je velik broj nepušača. Evo zagonetke:

Postoji pet kuća različitih boja u redu. U svakoj živi jedna osoba drugačije nacionalnosti. Pet vlasnika ovih kuća pije posebno piće, jede posebno voće i ima ljubimca. Ni jedan od njih ne jede isto voće, pije isto piće i ima istog ljubimca. Poznate činjenice su:

1. *Britanac živi u crvenoj kući.*
2. *Šveđanin ima psa za ljubimca.*
3. *Danac pije čaj.*
4. *Zelena kuća nalazi se s lijeve strane bijele kuće.*
5. *Vlasnik zelene kuće pije kavu.*

6. Vlasnik koji jede jabuke ima ptice za ljubimce.
7. Stanovnik žute kuće jede kruške.
8. Osoba koja živi u kući u sredini pije mlijeko.
9. Norvežanin živi u prvoj kući.
10. Vlasnik koji jede šljive živi pokraj kuće čiji vlasnik ima mačku.
11. Osoba koja ima konja živi pokraj osobe koja jede kruške.
12. Osoba koja jede lubenice pije pivo.
13. Nijemac jede trešnje.
14. Norvežanin živi pokraj plave kuće.
15. Osoba koja jede šljive živi pokraj čovjeka koji pije vodu.

Pitanje je: Tko ima ribicu za ljubimca?

Neke od ovih tvrdnji su jednostavne izjave, a neke su malo složenije. Postoji puno načina za rješavanje zagonetke[111]. Ovdje ćemo prikazati jednu varijantu rješenja u Prologu[112] koja je možda najbliže standardnim notacijama predikatne logike.

```
% Einsteinova zagonetka
% definiranje procedura
postoji(A, list(A, _, _, _, _)).
postoji(A, list(_, A, _, _, _)).
postoji(A, list(_, _, A, _, _)).
postoji(A, list(_, _, _, A, _)).
postoji(A, list(_, _, _, _, A)).

desnoOd(R, L, list(L, R, _, _, _)).
desnoOd(R, L, list(_, L, R, _, _)).
desnoOd(R, L, list(_, _, L, R, _)).
desnoOd(R, L, list(_, _, _, L, R)).

sredina(A, list(_, _, A, _, _)).

prva(A, list(A, _, _, _, _)).

nextTo(A, B, list(B, A, _, _, _)).
nextTo(A, B, list(_, B, A, _, _)).
nextTo(A, B, list(_, _, B, A, _)).
nextTo(A, B, list(_, _, _, B, A)).
nextTo(A, B, list(A, B, _, _, _)).
nextTo(A, B, list(_, A, B, _, _)).
nextTo(A, B, list(_, _, A, B, _)).
nextTo(A, B, list(_, _, _, A, B)).
```

[111] https://rosettacode.org/wiki/Zebra_puzzle

[112] Prolog verzija je inspirana rješenjem s https://curiosity-driven.org/prolog-interpreter .

```
% kodiranje tvrdnji
zagonetka(Kuce):-
  postoji(kuca(crvena, britanac, _, _, _), Kuce),
  postoji(kuca(_, svedjanin, _, _, pas), Kuce),
  postoji(kuca(zelena, _, kava, _, _), Kuce),
  postoji(kuca(_, danac, caj, _, _), Kuce),
  desnoOd(kuca(bijela, _, _, _, _), kuca(zelena, _, _, _, _), Kuce),
  postoji(kuca(_, _, _, jabuka, ptica), Kuce),
  postoji(kuca(zuta, _, _, kruska, _), Kuce),
  sredina(kuca(_, _, mlijeko, _, _), Kuce),
  prva(kuca(_, norvezanin, _, _, _), Kuce),
  nextTo(kuca(_, _, _, sljiva, _), kuca(_, _, _, _, macka), Kuce),
  nextTo(kuca(_, _, _, kruska, _),kuca(_, _, _, _, konj), Kuce),
  postoji(kuca(_, _, pivo, lubenica, _), Kuce),
  postoji(kuca(_, nijemac, _, tresnja, _), Kuce),
  nextTo(kuca(_, norvezanin, _, _, _), kuca(plava, _, _, _, _), Kuce),
  nextTo(kuca(_, _, _, sljiva, _), kuca(_, _, voda, _, _), Kuce),
  postoji(kuca(_, _, _, _, ribica), Kuce).

rjesenje(VlasnikRibice):-
  zagonetka(Kuce),
  postoji(kuca(_, VlasnikRibice, _, _, ribica), Kuce).
```

Na primjer prvu tvrdnju *„Britanac živi u crvenoj kući."* smo definirali izrazom:

`postoji(kuca(crvena, britanac, _, _, _), Kuce)`

gdje je `postoji(.,.)` unaprijed definirana procedura, a `kuca(_, _, _, _, _)` predikat kojim povezujemo boju, vlasnika, piće, voće i ljubimca.

Postavimo li upit:

`? - rjesenje(VlasnikRibice)`

dobit ćemo odgovor

`VlasnikRibice` `= nijemac`

Cijelu listu tko u kakvoj kući živi daje:

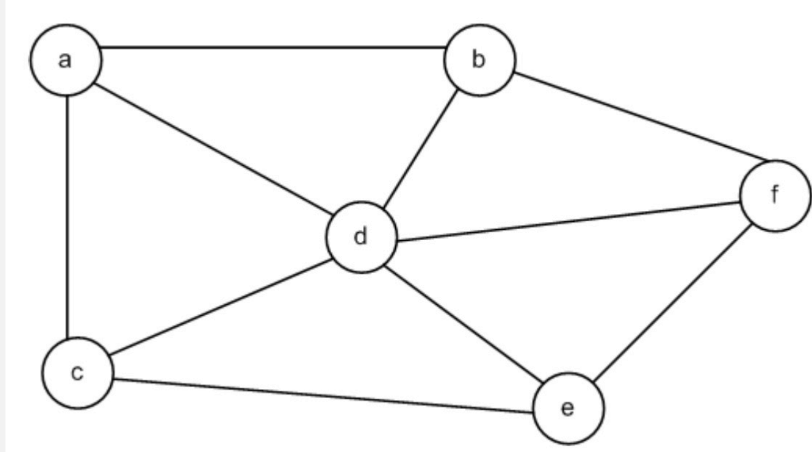
`? - zagonetka(Kuce)`

a odgovor glasi:

list(*kuca*(zuta, norvezanin, voda, kruska, macka), *kuca*(plava, danac, caj, sljiva, konj), *kuca*(crvena, britanac, mlijeko, jabuka, ptica), *kuca*(zelena, nijemac, kava, tresnja, ribica), *kuca*(bijela, svedjanin, pivo, lubenica, pas))

V3.6 Zadaci

Zadatak 3.6.1. Koristeći predikat `spojen` prikazati graf dan na slici V3-1, te postaviti upit kojim će se ispisati s kojim je sve čvorovima povezan čvor `e`.



Slika V3-1. Graf vezan uz zadatak 1

Primjer označavanja povezanosti čvora `a` s čvorom `b` (ali ne i čvora `b` s čvorom `a`): `spojen(a,b)`.

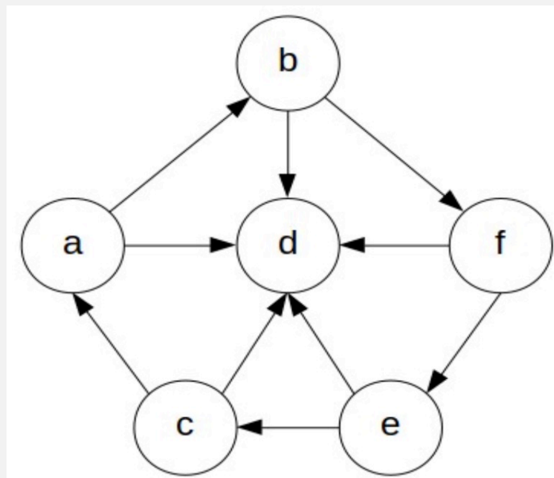
Zadatak 3.6.2. Napisati pravilo za predikat `povezan` kojim kažemo da nam nije bitan smjer u kojemu su povezani čvorovi grafa na slici V3-1: `a` je spojen s `b` ako je `b` spojen s `a`. Postaviti upit kojim će se ispisati s kojim je sve čvorovima povezan čvor `e`.

Zadatak 3.6.3. Definirati predikat `putanja` kojim pronalazimo putanje od jednog čvora do drugog. Definirati rekurzivno pravilo kojim kažemo:

postoji putanja od čvora do samog sebe,

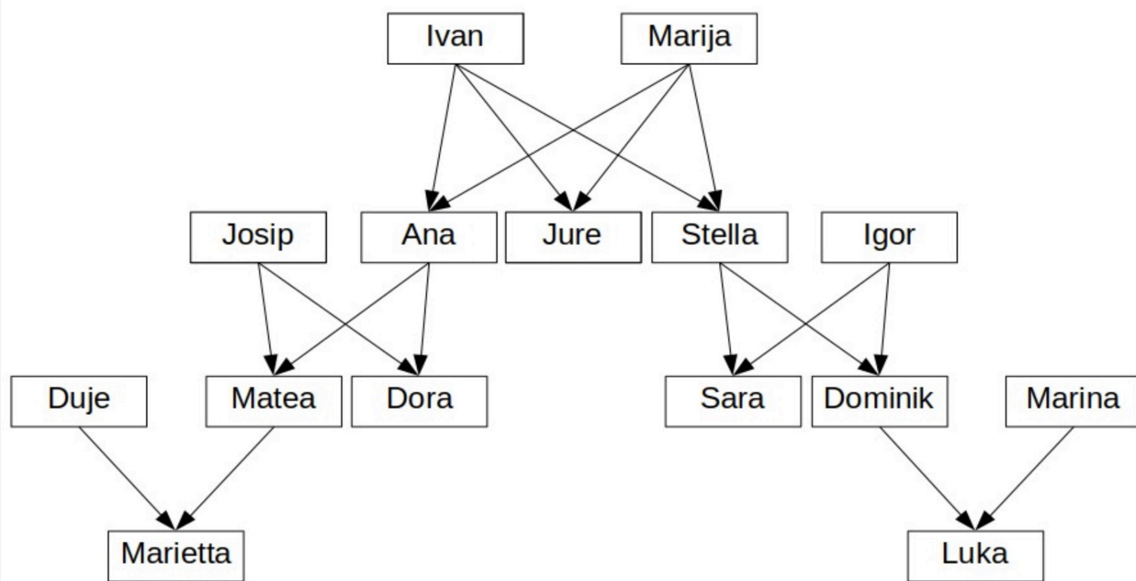
postoji putanja od čvora do čvora s kojim je spojem,

postoji putanja od čvora `a` do čvora `c` ako je `a` spojen sa čvorom `b` od kojega postoji putanja do `c`.



Slika V3-2. Graf vezan uz zadatak 3

Zadatak 3.6.4. Napisati bazu za obiteljsko stablo prikazano na Slici V3-4 koristeći predikate `majka` i `otac`. Napisati pravila za predikate `roditelj` i `baka`.



Slika V3-4. Obiteljsko stablo vezano uz zadatak 4

VJEŽBA 4 - UVOD U PYTHON

Python je interpretirani programski jezik koji je kreirao 1991. g. nizozemski programer ***Guido van Rossum*** (Google, Dropbox). Ime je dobio po seriji *„Monthy Python's Flying Circus"*. Sintaksa mu je slična C-u, što znači da je jednostavan za korištenje i može ga se lako naučiti[113]. Python je danas jako popularan među programerima koji se bave umjetnom inteligencijom jer je jako jednostavan i jer za njega postoji mnoštvo biblioteka vezanih za umjetnu inteligenciju.

Programi napisani u Pythonu imaju ekstenziju .py, a pokreću se s ***python ime_programa.py*** ili ***python3 ime_programa.py***, ovisno o instaliranoj verziji Pythona. Python ima mnoštvo raznih biblioteka, a one koje se najčešće koriste u strojnom učenju su:

- ***NumPy*** - za znanstvene proračune
- ***SciPy*** - za znanstvene proračune
- ***Scikit-learn*** - za strojno učenje
- ***Pandas*** - za obradu podataka
- ***OpenCV*** – za digitalnu obradu i analizu slike
- ***Pygame*** – za izradu računalnih igara.

Python podržava dva načina unosa komentara. U prvome načinu komentari započinju znakom #, a u drugom načinu komentari su omeđeni znakovima '' i "" koji se koristi u slučajevima kada je potrebno brzo komentirati više linija koda.

Primjeri komentara u Pythonu:

```
# ovo je komentar
"" ovo je komentar koji se
proteže u dvije linije ""
```

[113] Dobra škola za *on-line* učenje Pythona je: https://www.w3schools.com/python/python_intro.asp

Primjer ispisa znakovnog niza u Pythonu:

```
print ("Pozdrav!") # Python podržava dvostruke navodnike.
print ('Pozdrav!') # Python podržava jednostruke navodnike.
```

Za razliku od većine programskih jezika, Python ne sadrži vitičaste zagrade za označavanje pripadnosti dijela koda nekom uvjetu, petlji ili funkciji. Umjesto vitičastih zagrada u Pythonu se koristi tzv. uvlačenje linija (tabovi). Sve linije koda koje su uvučene ispod određenog uvjeta, petlje ili funkcije pripadaju tom uvjetu, petlji ili funkciji.

Primjer petlje koja ispisuje brojeve od 0 do 9:

```
for (i in range (0, 10)):
print (i)
```

Važno je imati na umu da Python razlikuje ***razmake*** (engl. *Space)* i tabove, te je u programskom kodu potrebno konzistentno koristiti tabove za uvlačenje linija koda, ili pak konzistentno razmake. Često se dogodi da početnici u Pythonu malo koriste tabove a malo razmake u istome programu, a to naposljetku rezultira greškom. ***Važno je zapamtiti da jedan tab ili određeni broj razmaka koji ukupno odgovaraju veličini tog taba u Pythonu nije jednako!***

U Pythonu se ne označavaju tipovi varijabli, već Python na temelju vrijednosti koja se dodijeli varijabli sam zaključi kojem tipu podataka ona pripada (npr. je li to int, float, string, itd.). Na primjer:

```
a = 10.0
b = 2
c = a / b
print("Vrijednost varijable c je " + str(c)) # c = 5.0
```

U gornjem primjeru možemo vidjeti da je Python sam zaključio da je varijabla *c* realni broj, a ne cijeli. Nadalje, možemo vidjeti i da se unutar funkcije *print* za ispis brojeva koristi funkcija *str* koja brojeve pretvara u znakovne nizove (tj. *stringove*), dok znak '+' vrši spajanje znakovnih nizova (tj. konkatenaciju).

U Pythonu se znakovi s tipkovnice koje korisnik unosi čitaju pomoću funkcije ***input()***. Na primjer:

```
print("Unesi neki broj: ")
broj = int(input()) # Pretvaramo string u broj.
if (broj > 10):
print("Broj je veći od 10!")
```

Primjer jednostavnog programa u Pythonu:

```
# ovo je komentar
print ("Kako se zoves?")
ime = input()
print ("Bok, " + ime + "!")
print (ime + ", koji je tvoj najdrazi grad?")
grad = input()
if (grad == "Split"):
print ("U tome se gradu trenutno nalazimo!")
elif (grad == "Zagreb"):
print (grad + " je glavni grad Hrvatske!")
else:
print (grad + " je lijep grad!")
```

V4.1 Uvjetni izrazi

Python podržava mnoštvo uvjetnih izraza poput `'=='`, `'!='`, `'<'`, `'>'`, `'<='` i `'>='`, te `if ... elif ... else` naredbi koji se često koriste. Ovi se uvjetni izrazi koriste na sličan način kao i u programskim jezicima C, C++, C#, Java, itd.

Primjeri uvjetnih izraza u Pythonu:

```
a = 4
b = 6
c = -1
d = 0
e = 11
if ( (a == 4) and (b < 10) ): # Logičko I se u Pythonu piše 'and'
print(c)
elif ( (a > 4) or (b < 0) ): # Logičko ILI se u Pythonu piše 'or'
print(d)
if ( not(d > 0) ): # Logička negacija se u Pythonu piše 'not'
print(e)
else:
print(e)
```

V4.2 Naredbe ponavljanja

Python podržava mnoštvo naredbi ponavljanja poput ***while*** i ***for*** petlji koje se često koriste.

Primjer ***while*** petlje:

```
i = 0
while (i < 10):
print(i) # Ispisuju se brojevi od 0 do 9.
i = i + 1
```

Primjeri ***for*** petlje:

```
i = 0
for i in range(0,10):
print (i) # Ispisuju se brojevi od 0 do 9.
i = 0
for i in range(10):
print (i) # Ispisuju se brojevi od 0 do 9.
```

V4.3 Funkcije

Funkcije se u Pythonu definiraju pomoću ključne riječi ***def***. Na primjer:

```
def imeFunkcije():
print("Unesi neki string: ")
nekiString = input()
print("Unijeli ste slijedeci string: " + nekiString)
```

Ako funkcija u Pythonu treba vratiti neku vrijednost, onda se to radi pomoću naredbe ***return***. Na primjer:

```
def zbrajanje(a, b): # Funkcija zbrajanje prima argumente a i b.
c = a + b
return c # Funkcija vraća varijablu c.
```

V4.4 Nizovi i matrice

Nizove je u Pythonu jednostavno definirati. Na primjer:

```
niz1 = [1, 2, 3, 4, 5] # Niz brojeva.
niz2 = ["prvi", "drugi", "treci"] # Niz stringova.
print (niz1[0]) # Ispiši prvi element prvog niza.
print (len(niz2)) # Ispiši veličinu drugog niza.
```

U Pythonu se rad s matricama najjednostavnije obavlja korištenjem biblioteke ***numpy***. Na primjer:

```
import numpy as np # np ćemo koristiti kao skraćeni oblik zapisa za numpy.
matrica = np.zeros((3,3)) # Stvaramo matricu dimenzija 3x3 punu nula.
print(matrica) # Ispisujemo matricu.
matrica[0][0] = 1 # Mijenjamo vrijednost elementa matrice.
```

V4.5 Zadaci

Zadatak 4.5.1. Napisati program u Pythonu koji u ovisnosti o tome koji broj korisnik upiše provjerava je li taj broj bez ostatka djeljiv s 2.

Zadatak 4.5.2. Napiši program u Pythonu koji funkcionira kao jednostavni kalkulator za osnovne aritmetičke operacije zbrajanje, oduzimanje, množenje i dijeljenje.

Zadatak 4.5.3. Napiši program u Pythonu za množenje dviju matrica.

VJEŽBA 5 - STROJNO UČENJE U PYTHONU

Strojno učenje (engl. *Machine Learning)* je dio umjetne inteligencije koji proučava načine na koji programi mogu učiti i zaključivati. Postoje tri vrste strojnog učenja:

- ***nadzirano učenje*** (engl. *Supervised Learning*),
- ***nenadzirano učenje*** (engl. *Unsupervised Learning*),
- ***polunadzirano učenje*** (engl. *Semi-supervised Learning*) od kojeg je najpoznatije pojačano učenje (engl. *Reinforcement Learning*).

V5.1 Nadzirano učenje

Nadziranim učenjem pokušava se pronaći način kako, na temelju ulaznih podataka za koje je poznato kojim klasama pripadaju, klasificirati nove podatke. Nadzirano učenje obično uključuje bazu znanja i metodu zaključivanja. Problem nadziranog učenja je nemogućnost ispravnog postupanja s novim informacijama koje se nisu koristile u fazi treniranja.

Ekspertni ili stručni sustavi *(*engl. *Expert Systems)* često koriste metode nadziranog učenja jer nastoje emulirati znanje ljudskih stručnjaka u određenom području. Primjer jednostavnog ekspertnog sustava koji identificira jednu od sljedeće tri životinje - papiga, vrabac, riba:

```
print("Može li životinja letjeti?")
let = input()
if (let == "da"):
print("Može li životinja govoriti?")
govor = input()
if (govor == "da"):
print("Papiga")
else:
print("Vrabac")
else:
print("Riba")
```

Primjer upotrebe nadziranog učenja u ekspertnom sustavu je ***klasifikator stabla odluke*** *(*engl. *Decision Tree Classifier)*. Klasifikator stabla odluke temelji se na strukturi sličnoj dijagramu toka kod kojeg:

- unutarnji čvorovi predstavljaju značajku ili atribut
- grane predstavljaju pravila odluke, a
- krajnji čvorovi ishod odluke.

Primjer baze znanja s kojom ćemo trenirati klasifikator stabla odluke dan u tablici V5-1. Značajke na temelju kojih se odluka donosi su *vrijeme, temperatura* i *vjetar*, a ishod je binaran, utakmica će se igrati (*da*) ili neće (*ne*).

Tablica V5-1. Primjer baze znanja s kojom ćemo trenirati klasifikator stabla odluke

vrijeme	**temperatura**	**vjetar**	**utakmica ?**
oblačno	23 °C	umjeren	da
sunčano	27 °C	slab	da
oblačno	-10 °C	jak	ne
sunčano	-30 °C	slab	ne
kišno	18 °C	slab	ne
sunčano	0 °C	jak	ne

Kada se baze znanja koriste u praksi, mogu se u memoriju računala spremiti na mnoštvo različitih načina. Način koji se najčešće koristi u Pythonu je spremanje baze znanja u obliku ***CSV*** *(*engl. *Comma Separated Values)* formata. Na primjer, CSV oblik podataka prikazanih u tablici V-2 koji možemo nazvati *train_dataset.csv* bio bi:

vrijeme,temperatura,vjetar,utakmica

oblacno,23,umjeren,da

suncano,27,slab,da

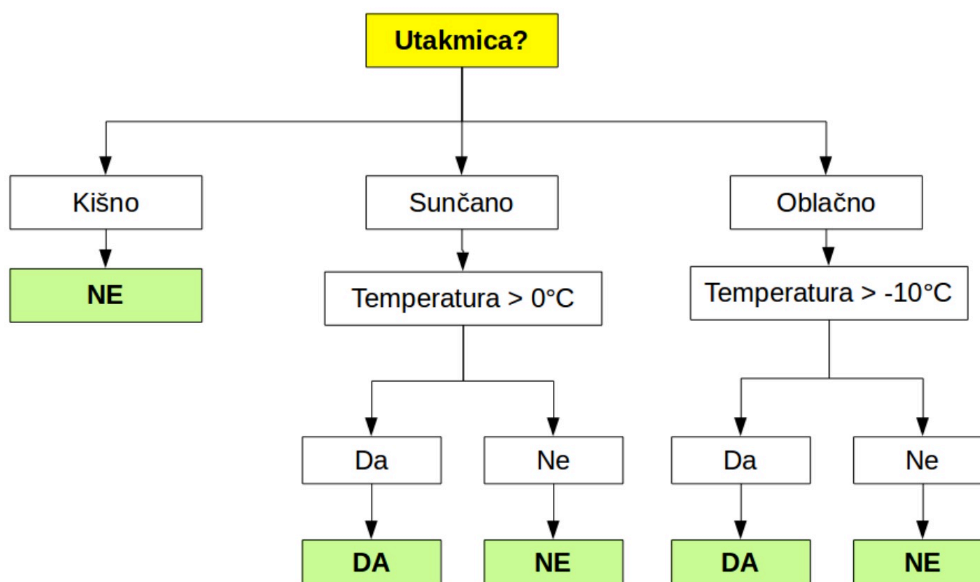
oblacno,-10,jak,ne

suncano,-30,slab,ne

kisno,18,slab,ne

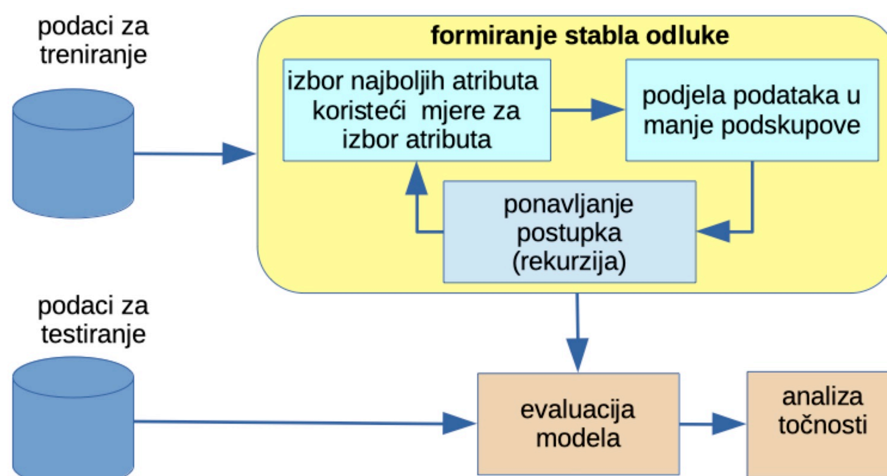
suncano,0,jak,ne

Stablo odluke za atribute *vrijeme* i *temperatura* prikazuje slika V5-1



Slika V5-1. Stablo odluka za podatke iz tablice V5-2

Postupak klasifikatora stabla odluke sastoji se od nekoliko koraka prikazanih na slici V5-2.



Slika V5-2. Grafički prikaz algoritma klasifikatora stabla odluke

Stablo odluke formira se na temelju podataka za treniranje danih u tablici V5-2.

Prvi korak je izbor najboljih atributa koristeći ***mjere za izbor atributa*** (engl. *ASM – Attribute Selection Measures)* koje predstavljaju heurističku mjeru na temelju koje se podaci dijele u manje podskupove na najbolji mogući način. Podijeljeni podaci rezultat ovog su postupka, Nakon toga kreće izgradnja stabla odluke ponavljajući ovaj proces za svaki podskup sve dok:

- nema više preostalih atributa
- nema više ishoda odluke ili
- sve dobivene n-torke (engl. *touples)* pripadaju istoj vrijednosti atributa.

Python u okviru biblioteke funkcija ***scikui-learn***[114] ima funkciju `DecisionTreeClassifier()`[115] kojom se realizira ovaj postupak. Funkcija ima tri parametra kojima je moguće optimirati postupak izgradnje stabla odluke. Prvi je odabir postupka za izbor atributa `criterion=".".` Dostupne su dvije mjere[116]:

- Ginijev koeficijent koji je zadana (*default*) vrijednost ili
- entropija (`criterion="entropy"`).

Ginijev koeficijent (engl. *Gini Index)* je statistička mjera za izračun neravnomjerne raspodjele, a uveo ga je još 1912. g. talijanski statističar ***Corrado Gini*** (1884.-1965.) u području ekonomije. Ginijev koeficijent razmatra binarnu podjelu svakog atributa, te računa otežanu sumu vjerojatnosti te podjele.

$$Gini(D) = 1 - \sum_{i=1}^{m} P_i^2$$

gdje je P_i vjerojatnost da n-torka iz D pripada jednoj od klasa ishoda odluke. Kako imamo binarnu podjelu, svaki od atributa A dijeli skup D na dva podskupa $D1$ i $D2$, a vrijednost Ginijevog koeficijenta ove podjele je:

$$Gini_A(D) = \frac{|D1|}{|D|} \cdot Gini(D1) + \frac{|D2|}{|D|} \cdot Gini(D2)$$

gdje je $|.|$ kardinalnost (broj elemenata) pojedinog skupa. Onaj atribut A koji ima najmanji Ginijev koeficijent bira se kao atribut na kojem će se raditi podjela kod izgradnje stabla odluke. Ako atribut ima kontinuiranu vrijednost (kao npr. *temperatura* u tablici V-2) uzima se par po par susjednih vrijednosti pa se računa razlika Ginijevih koeficijenata:

$$\Delta Gini(A) = Gini(D) - \cdot Gini_A(D)$$

te se odabire onaj za koji je ova razlika najmanja.

Entropija ili ***informacijsko pojačanje*** (engl. *Information Gain)* temelji se na proračunu razlike informacijske entropije prije dijeljenja podataka u podskupove i srednje entropije nakon djeljenja podataka u podskupove. Informacijska entropija nekog skupa podataka je na neki način mjera uređenosti tog skupa. Što je veće informacijsko pojačanje, to je i veća uređenost skupa i bolja podjela.

$$Info(D) = 1 - \sum_{i=1}^{m} P_i \cdot log_2 P_i$$

gdje je P_i isto kao prije vjerojatnost da n-torka iz D pripada jednoj od klasa ishoda odluke.

$$Info_A(D) = \sum_{i=1}^{V} \frac{|Dj|}{|D|} \cdot Info(Dj)$$

Informacijsko pojačanje za podjelu po atributu A računa se izrazom:

$$Gain(A) = Info(D) - \cdot Info_A(D)$$

[114] https://scikit-learn.org/stable/index.html

[115] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[116] Više detalja o mjerama za izbor atributa u http://www.ijoart.org/docs/Construction-of-Decision-Tree--Attribute-Selection-Measures.pdf

Bira se onaj atribut koji ima najveće informacijsko pojačanje.

Kad je postupak gotov treba nam i skup podataka za testiranje. U našem primjeru to je `test_dataset.csv` kojeg formiramo iz tablice V5-2:

Tablica V5-2. Podaci za testiranje klasifikatora stabla odluke treniranog podacima iz tablice V5-1

vrijeme	temperatura	vjetar	utakmica?
kišno	3 °C	umjeren	ne
kišno	27 °C	slab	ne
oblačno	10 °C	slab	da
sunčano	-1 °C	slab	da
kišno	20 °C	jak	ne
sunčano	0 °C	umjeren	da

Za rješenje ovog zadatka trebaju sljedeće Python biblioteke:

- *csv* – Comma Separated Value
- *pandas* – Python Data Analysis Library – PIL --> Image – Python Imaging Library
- *sklearn.metrics* --> *accuracy_score* – analiza točnosti
- *PIL* --> *Image* – Python Imaging Library
- *sklearn* --> *tree* – stabla odluke
- *numpy* – za znanstvene proračune matplotlib.pyplot – za crtanje 2D grafova
- *sklearn.preprocessing* --> *LabelEncoder* – za mijenjanje kategoričnih vrijednosti iz baze znanja u numeričke

Cijeli Python kod postupka je:

```
import csv
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn import tree
import numpy as np
from sklearn.preprocessing import LabelEncoder

#inicijaliziraj LabelEncoder
le = LabelEncoder()

# ucitaj csv dokument za treniranje
print("\n Podaci za treniranje")
```

```
df_train = pd.read_csv('train_dataset.csv', sep = ',',header = 0)
print(df_train)

# ucitaj csv dokument za testiranje
print("\n Podaci za testiranje")
df_test = pd.read_csv('test_dataset.csv', sep = ',',header = 0)
print(df_test)

# zamjena kategoričkih varijabli numeričkim
for col in df_test.columns.values:
   if df_test[col].dtypes == 'object':
         data = df_train[col].append(df_test[col])
         le.fit(data.values)
         df_train[col] = le.transform(df_train[col])
         df_test[col] = le.transform(df_test[col])

# podaci za treniranje
print("Podaci za treniranje:")
print(df_train)
print("\n")

# podaci za testiranje
print("Podaci za testiranje:")
print(df_test)
print("\n")

# podjela podataka u vektore atributa(X) i ishode(y)
X_train, y_train = df_train.iloc[:,:-1], df_train.iloc[:, -1]
X_test, y_test = df_test.iloc[:,:-1], df_test.iloc[:, -1]

print("Podaci:")
print(X_train)
print("\n")
print("Labele:")
print(y_train)
print("\n")

# klasifikacija
clf = tree.DecisionTreeClassifier()
```

```
clf.fit(X_train, y_train)

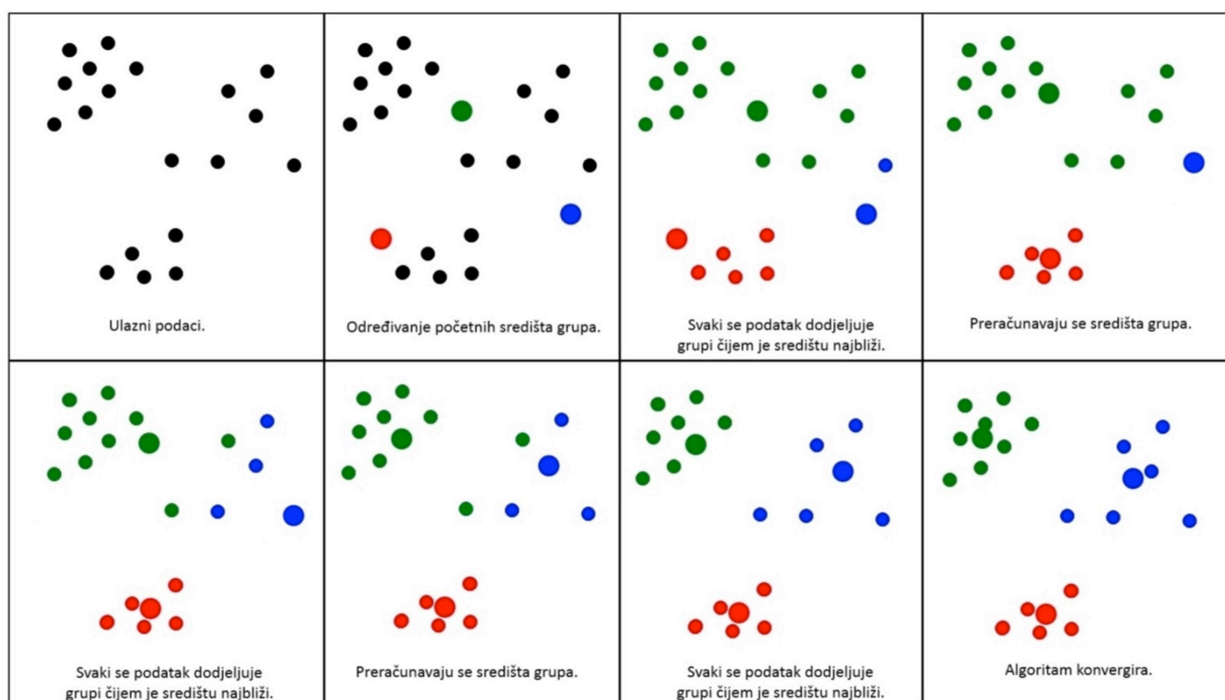
# tocnost na train setu
print(clf.score(X_train, y_train))
predicted = clf.predict(X_test)
print(predicted)
print("\n")
print ("Tocnost je ", accuracy_score(y_test,predicted)*100)
print("\n")
```

V5.2 Nenadzirano učenje

Nenadzirano učenje koristi se za pronalazak strukture u podacima za koje ne znamo kojoj kategoriji pripadaju. Primjeri primjene nenadziranog učenja su npr. analiza piksela na slici u svrhu određivanja postojećih klasa boje na slici ili grupiranje pacijenata koji su imali jednu bolest s nekom drugom bolešću.

Primjer algoritma nenadziranog učenja je ***algoritam k-sredine*** (engl. *k-means Algorithm)* kojim se provodi klasteriranje u n klasa. Shematski prikaz rada ovog algoritma dan je na slici V5-3 gdje su prikazani koraci klasteriranja podataka u 3 klase.

Python u okviru biblioteke funkcija ***scikit-learn*** ima funkciju `KMeans()` za proračun algoritma k-sredine[117].



Slika V5-3. Shematski prikaz rada algoritma k-sredina

[117] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

V5.3 Pojačano učenje kao primjer polunadziranog učenja

Polunadzirano učenje sadrži karakteristike i nadziranog i nenadziranog učenja. Ovakav način učenja može se koristiti prilikom konstrukcije algoritama za klasifikaciju djelomično označenih podataka. Na primjer pretpostavimo da radimo klasifikator kojem je zadatak prepoznavati slike automobila pri čemu su tijekom faze treniranja korišteni podaci od kojih je samo jedan manji dio označen (npr. od 1000 slika algoritmu je kazano da 200 slika sigurno sadržava automobile, a za preostalih 800 slika algoritam ne dobiva informacije o tome sadrže li automobile ili ne).

Pojačano učenje je primjer algoritma polunadziranog učenja koje se temelji na sustavu nagrada i kazni. Obično se koristi u agentskim sustavima. Primjer upotrebe pojačanog učenja je konstrukcija automatskih igrača u računalnim igrama tzv. NPC-ova od engl. *Non-Player Character*. U programskom Python alatu ***OpenAI Gym***[118] nalaze se sve funkcije za razvoj i usporedbu različitih postupaka pojačanog učenja, na primjer implementaciju Q-učenja.

V5.4 Zadaci

Zadatak 5.4.1. Napišite jednostavni ekspertni sustav koji pogađa na koju vrstu vode (jezero, more, potok, rijeka) korisnik misli.

Zadatak 5.4.2. Primjeni Python program klasifikatora stabla odluke za datoteku o autentičnosti novčanica iz repozitorija datoteka strojnog učenja University of California[119]. Autentičnost novčanica predviđa se na temelju četiri atributa koji su izvučeni sa slike novčanice i pripadaju području napredne analize digitalne slike: ***varijanca wavelet transformirane slike, kurtozis slike, entropija i nakrivljenost slike*** (engl. *variance of wavelet transformed image, curtosis of the image, entropy, and skewness of the image).* U zadnjoj koloni je binarna informacija koja kaže je li novčanica lažna (0) ili prava (1). Podatke podijeli u dvije skupine (80% lažnih i pravih za treniranje i ostatak za testiranje).

Zadatak 5.4.3. Napiši program u Pythonu za klasteriranje u 3 klase vina algoritmom k-sredine iz datoteke vina s repozitorija datoteka strojnog učenja University of California[120] koja sadrži podatke ispitivanja 178 vina iz jedne talijanske regije. Klasteriranje provedi u dvodimenzionalnom prostoru na temelju varijabli 'alchocol' i 'OD280/OD315 of diluted wines'.

Zadatak 5.4.4. Napiši program u Pythonu koji pomoću Q-pojačanog učenja rješavanje labirint sa slike V5-4.

0 START	1	2	3	4
5	6	7	8	9
10	11	12	13	14 CILJ

Slika V5-4. Labirint koji treba riješiti u zadatku 4 (pojačana linija je zid, što znači da se npr. ne može ići iz 1 u 6)

[118] https://gym.openai.com

[119] https://archive.ics.uci.edu/ml/datasets/banknote+authentication – za repozitorij su zaslužni: Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[120] https://archive.ics.uci.edu/ml/datasets/Wine - za repozitorij su zaslužni autori iz fusnote 107.

Dozvoljena su samo horizontalna i vertikalna kretanja, a način nagrađivanja/kažnjavanja uzmite iz poglavlja 6.5.1. Kako izgleda konačna Q-matrica?

VJEŽBA 6 - LINEARNA REGRESIJA

Linearna regresija je metoda nadziranog strojnog učenja koja se koristi za predviđanje kontinuiranih varijabli linearnom hipotezom temeljem više vrijednosti (npr. predviđanje cijena kuća u ovisnosti o površini, lokaciji, orijentaciji, opremanju i sl.). U ovoj laboratorijskoj vježbi predviđat ćemo cijene određenih objekata na tržištu na temelju njihovih značajki. Python u okviru biblioteke funkcija ***scikit-learn*** i modula LinearRegression ima funkciju `LinearRegression()` za proračun linearne regresije[121].

V6.1 Zadaci

Zadatak 6.1.1. Prikupite podatke o cijenama određenih objekata koji su trenutno na tržištu te o značajkama tih objekata. Smjernice:

- Što više informacija prikupite, algoritam koji ćete razviti u nastavku laboratorijske vježbe bit će točniji. Prikupite najmanje 40 podataka.
- Ako ne pronađete određenu informaciju za neki objekt, označite taj podatak upitnikom.
- Podaci koje trebate prikupiti ovise o grupi laboratorijskih vježbi u kojoj odrađujete vježbu na način prikazan u tablici V6-1.

Tablica V6-1. Teme po grupama

grupa	tema	primjer
1	predviđanje cijene nekretnina	tablica V6-2
2	predviđanje cijene rabljenih automobila	tablica V6-3
3	predviđanje cijene rabljenih motocikla	tablica V6-4
4	predviđanje cijene novih prijenosnih računala	tablica V6-5
5	predviđanje cijene rabljenih plovila	tablica V6-6
6	predviđanje cijene novih mobitela	tablica V6-7

Tablica V6-2 Primjer podataka vezanih uz prodaju nekretnina

cijena (€)	površina (m²)	lokacija	kat	dizalo	privatni parking	balkon
600000	130	Split (Meje)	3	Ne	Da	Da
300000	65	Split (Centar)	2	Ne	Ne	Da
60000	30	Split (Radunica)	1	Ne	Ne	Ne

[121] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression

Tablica V6-3. Primjer podataka vezanih uz prodaju rabljenih automobila

cijena (€)	marka	model	u prometu od	prijeđeni kilometri
7000	Alfa Romeo	Giulietta	2010.	130000
18000	Audi	A6 Allroad	2010.	140000
15000	Jeep	Commander	2012.	250000

Tablica V6-4. Primjer podataka vezanih uz prodaju rabljenih motocikla

cijena (€)	marka	model	u prometu od	prijeđeni kilometri
12000	Ducati	Multistrada 950	2017.	9000
7000	Suzuki	Gsx-r 600	2010.	100000
1500	Piaggio	Vespa	2000.	100000

Tablica V6-5. Primjer podataka vezanih uz nova prijenosna računala na tržištu

cijena (kn)	proizvođač	operacijski sustav	procesor	RAM (GB)
22000	ASUS	Windows 10	Intel Core i7 8750H	8
6000	DELL	Linux	Intel Core i5 8300H	8
9000	HP	Windows 10	Intel Core i7 7700HQ	8

Tablica V6-6. Primjer podataka vezanih uz prodaju rabljenih plovila

cijena (kn)	proizvođač	tip plovila	duljina (m)	godina proizvodnje
85000	Bavaria	Jedrilica	16	2003.
18000	Barracuda 500	Gumenjak	5	2014.
2500	Lasta Vodice	Čamac	6	1990.

Tablica V6-7. Primjer podataka vezanih uz prodaju novih mobitela

cijena (kn)	proizvođač	model	kamera (MP)	memorija (GB)
500	Apple	iPhone 8	12	64
700	Apple	iPhone 8	12	256
600	Samsung	Galaxy Note 9	12	512

Zadatak 6.1.2. Podatke prebacite u CSV (engl. *Comma Separated Value*) format i sačuvajte kao datoteku s ekstenzijom .csv. Primjer CSV datoteke:

```
Cijena,Povrsina,Lokacija,Kat,Dizalo,Privatni_parking,Balkon
600000,130,Split (Meje),3,Ne,Da,Da
300000,65,Split (Centar),2,Ne,Ne,Da
60000,30,Split (Radunica),1,Ne,Ne,Ne
```

Zadatak 6.1.3. CSV datoteku iz prethodnog zadatka podijelite na dva dijela - pola podataka spremite u datoteku `train.csv` (na ovim ćemo podacima trenirati algoritam) a pola u `test.csv` (na ovim ćemo podacima testirati algoritam).

Zadatak 6.1.4. Napišite program u Pythonu koji će pomoću biblioteke ***scikit-learn*** (i modula ***LinearRegression***) predviđati cijene objekta zadanog u zadatku 1. Primjer programa za klasifikaciju podataka pomoću ***scikit-learn*** biblioteke možete pronaći u primjerima vježbe 5.

VJEŽBA 7 – NAIVNI BAYESOV KLASIFIKATOR

Naivni Bayesov klasifikator (engl. *Naive Bayes Classifier)* jedan je od najjednostavnijih klasifikatora u strojnom učenju, i često daje dosta dobre rezultate. Budući da ne zahtijeva složena računanja, jako je brz u usporedbi s drugim klasifikacijskim algoritmima. Temelji se na Bayesovom teoremu:

$p(a|b)=p(b|a) \cdot p(a) / p(b)$

gdje su:

$p(a|b)$ – vjerojatnost da se dogodi događaj a ako se je dogodio događaj b

$p(a)$ – vjerojatnost da se dogodi događaj a

$p(b|a)$ – vjerojatnost da se dogodi događaj b ako se je dogodio događaj a

$p(b)$ – vjerojatnost da se dogodi događaj b.

Ime *„naivni"* dobio je zato što se u radu ovog klasifikatora pretpostavlja da su sve varijable u ulaznim podacima neovisne jedna o drugoj, što vrlo često nije istina. U formuli za proračun $p(a|b)$ dio $p(b)$ se često može zanemariti. Primjer naivnog Bayesovog klasifikatora za ulazne podatke s više varijabli možete pronaći na linku: https://www.youtube.com/watch?v=ZAfarappAO0.

Tablica V7-1 Primjer podataka za klasifikaciju

kolegij	ocjena
matematika	4
matematika	3
matematika	5
matematika	1
matematika	4
matematika	4
fizika	4
fizika	5
fizika	5
fizika	3

Tablica V7-2 Tablica koja prikazuje koliko puta se je određena ocjena pojavila za određeni kolegij

kolegij	ocjena: 1	ocjena: 2	ocjena: 3	ocjena: 4	ocjena: 5
matematika	1	0	1	3	1
fizika	0	0	1	1	2

Računamo vjerojatnost da se u ulaznim podacima pojavi kolegij *matematika*:

p(matematika)= 6 / 10 = 0,6

Računamo vjerojatnost da se u ulaznim podacima pojavi kolegij *fizika*:

p(fizika)= 4 / 10 = 0,4

Računamo vjerojatnost da se u ulaznim podacima pojavi *ocjena 1*:

p(ocjena1)= 1/ 10 = 0,1

Računamo vjerojatnost da se u ulaznim podacima pojavi *ocjena 2*:

p(ocjena2)= 0 / 10 = 0

Računamo vjerojatnost da se u ulaznim podacima pojavi *ocjena 3*:

p(ocjena3)= 2/10 = 0,2

Računamo vjerojatnost da se u ulaznim podacima pojavi *ocjena 4*:

p(ocjena4)= 4 / 10 = 0,4

Računamo vjerojatnost da se u ulaznim podacima pojavi *ocjena 5*:

p(ocjena5)= 3/ 10 = 0,3

Primjer računanja vjerojatnosti da iz kolegija *matematika* bude *ocjena 5*:

$p(ocjena5 \mid matematika) = p(matematika \mid ocjena5) \cdot p(ocjena5) / p(matematika) =$

$= (1/6) \cdot 0{,}3 / 0{,}6 = 0{,}0833$

Primjer Python programa za računanja vjerojatnosti da se određeni kolegij pojavi u ulaznim podacima koji su sačuvani u datoteci s ekstenzijom .csv (i imaju oblik *ime_kolegija,ocjena*):

```
import csv
import pandas as pd import numpy as np
# ucitaj CSV dokument za treniranje
print("Podaci za treniranje")
df_train = pd.read_csv('train.csv', sep = ',', header = 0) print(df_train)

# izdvajanje stupaca iz ulaznih podataka stupac1_train = df_train.iloc[:, 0]
stupac2_train = df_train.iloc[:, 1]
br_matematika = 0 br_fizika = 0
for i in range (0, stupac1_train.size):
if (stupac1_train[i] == "Matematika"):
br_matematika = br_matematika + 1 if (stupac1_train[i] == "Fizika"):
br_fizika = br_fizika + 1
p_matematika = 0
p_fizika = 0
p_matematika = float(p\_matematika) p_fizika = float(p\_fizika)
# vjerojatnost da se u ulaznim podacima pojavi matematika p_matematika = br_matematika
/ (br_matematika + br_fizika)
# vjerojatnost da se u ulaznim podacima pojavi fizika p_fizika = br_fizika /
(br_matematika + br_fizika)
```

V7.1 Zadaci

Zadatak 7.1.1. Prikupite proizvoljne ulazne podatke koje ćete koristiti prilikom treniranja i testiranja ***naivnog Bayesovog klasifikatora***.

Zadatak 7.1.2. Za proizvoljne ulazne podatke koje ste prikupili u 1. zadatku konstruirajte naivni Bayesov klasifikator u Pythonu. Nemojte koristiti ugrađenu funkciju za ovaj klasifikator, već napišite sami svoju (tj. napišite funkciju koja će sama računati vjerojatnosti kao što su vjerojatnost da se u ulaznim podacima pojavi kolegij matematika, vjerojatnost da se u ulaznim podacima pojavi kolegij fizika, vjerojatnost da se za kolegij fizika pojavi ocjena 5 itd.).

Zadatak 7.1.3. Usporedite rezultate koje ste dobili s rezultatima koje daje ugrađena funkcija za naivni Bayesov klasifikator iz ***scikit-learn*** biblioteke u Pythonu.

VJEŽBA 8 – UMJETNE NEURONSKE MREŽE U PYTHONU

Umjetne neuronske mreže (engl. *ANN – Artificial Neural Networks*) dio su strojnog učenja koji zadatke rješava imitiranjem biološke neuralne mreže. Iako se pojmovi *„umjetne neuronske mreže"* i *„neuronske mreže"* razlikuju, u računarstvu se često koriste kao sinonimi, tj. riječ *„umjetne"* obično se ne naglašava, nego se podrazumijeva.

Duboko učenje (engl. *Deep Learning)* je naprednija verzija neuronskih mreža. Riječ „duboko" samo naglašava da ova metoda strojnog učenja koristi više razina međuslojeva, tj. dublja je od uobičajenih neuronskih mreža.

Najčešće primjene neuronskih mreža su:

- klasifikacija
  - OCR (engl. *Optical Character Recognition*) sustavi za automatsko prepoznavanje teksta
  - klasifikacija digitalnih slika
- klasteriranje
  - klasifikacija bez poznatih klasa (nenadzirano učenje).

V8.1 Perceptron

Perceptron je najjednostavniji oblik umjetnih neuronskih mreža koje modeliraju biološke neurone. Više međusobno povezanih perceptrona čini umjetnu neuronsku mrežu koja nakon treniranja može donositi različite odluke i zaključke, te rješavati složene probleme. Perceptron može imati više ulaznih podataka, ali samo jedan izlazni. I ulazni i izlazni podaci perceptrona predstavljaju se pomoću binarnih brojeva. Svaki ulaz u perceptron ima svoju težinu. Ta težina može biti prethodno definirana ili je neki od algoritama strojnog učenja može automatski izračunati. Shematski prikaz perceptrona prikazuje slika 6-16.

Zadatak 8.1.1. Kako pomoću jednostavnog perceptrona odlučiti hoćete li ići s prijateljima u kazalište ili ne? Opis ovog perceptrona prikazuje slika V8-1.

Upute za izradu odgovarajućeg perceptrona:

- predstavite svaki faktor koji utječe na vašu odluku kao ulaz u perceptron
- odredite težine svakog faktora na temelju toga koliko su vam ti faktori važni
- proučite odgovore na vaše uvjete i zbrojite njihove težine

- definirajte neki prag (engl. *Threshold)* – ako je zbroj težina veći od tog praga, ići ćete u kazalište, a ako nije, onda nećete. O definiranom pragu ovisi rad perceptrona.

x_1: Hoće li vrijeme biti sunčano?
w_1 = 2 ako će vrijeme sunčano
w_1 = 0 ako vrijeme neće biti sunčano

x_2: Hoće li ići moj najbolji prijatelj ili prijateljica?
w_2 = 5 ako će ići
w_2 = 3 ako neće ići

x_3: Hoću li premašiti određeni budžet?
w_3 = 0 ako hoću
w_3 = 4 ako neću

x_4: Zanima li me predstava?
w_4 = 7 ako me zanima
w_4 = 4 ako me ne zanima

w_5: Koliko je kazalište daleko od moje kuće?
w_5 = 0 ako je daleko
w_5 = 4 ako nije daleko

x_1: Oblačno
x_2: Najbolji prijatelj ide
x_3: Premašiti ću budžet
x_4: Zanima me predstava
x_5: Nije daleko

w_1: 0
w_2: 5
w_3: 0
w_4: 7
w_5: 4

$Y = w_1 + w_2 + w_3 + w_4 + w_5 = 16$

Ako je Y > threshold
izlaz = 1
Ako je Y <= threshold
izlaz = –1

Slika V8-1. Opis perceptrona koji odlučuje o odlasku u kazalište

V8.2 Sigmoidni umjetni neuron

Perceptroni uče na način da težine w_1 ... w_n pokušavaju podesiti tako da krajnja odluka perceptrona bude što točnija. Budući da mala promjena u težini jednog faktora može uzrokovati velike promjene u ponašanju cijelog sustava, umjesto perceptrona koriste se sigmoidni neuroni. Sigmoidni neuroni se od perceptrona razlikuju po tom što:

- mala promjena u težini nekog od ulaznih faktora uzrokuje male promjene u cijelom sustavu
- izlaz sigmoidnog neurona može poprimiti bilo koju vrijednost iz intervala [-1, 1], a ne samo -1 ili 1 kao kod perceptrona.

V8.2.1 Aktivacijska funkcija

Aktivacijska funkcija je krivulja u obliku slova "S". Slična je sigmoidnoj funkciji iz logističke regresije, samo što je spuštena tako da daje izlazne vrijednosti u intervalu [-1 , 1], a kreira se na primjer pomoću ***hiperbolnog tangesa***:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Aktivacijska funkcija će broj koji dobije kao ulazni podatak prebaciti u broj iz intervala [-1, 1]. Njezin zadatak je uvođenje nelinearnosti u neuralnu mrežu jer su ulazni podaci najčešće nelinearni.

Python kod za aktivacijsku funkciju i njenu derivaciju:

```
import numpy
def izracunaj_sigmoid(x)
y = (numpy.exp(x) - numpy.exp(-x)) / (numpy.exp(x) + numpy.exp(-x))
```

```
return y
def izracunaj_sigmoid_derivaciju(s)
y = 1 - s * s
return y
```

Osim ove aktivacijske funkcije koriste se i neke druge funkcije, a danas posebno ***ispravljena linearna funkcija*** (engl. *ReLU – Rectified Linear Unit*) i njena modificirana verzija ***parametarska ReLU*** (engl. *Parametric ReLU)*:

$$f(x) = \begin{cases} \alpha x, & za\ x \leq 0 \\ x, & za\ x > 0 \end{cases}$$

Ako je α = 0,01 naziva se ***propustljiva ReLU*** (engl. *Leaky ReLU).* U brojnim primjerima ove aktivacijske funkcije daju bolje rezultate. Njihove izlazne vrijednosti su u intervalu [-∞, +∞].

Izlaz sigmoidnog neurona s n ulaza je:

$$\mathrm{y} = S(\textstyle\sum_{i=1}^{n} w_i x_i + \mathrm{b})$$

gdje su:

S – aktivacijska funkcija

w_i – težina ulazne veličine

x_i – ulazna veličina

b – tzv. *bias* (tj. prag s obrnutim predznakom).

Treniranje neuronskih mreža je proces koji obuhvaća pronalazak težina w_i i praga b koji bi minimizirali određenu ***cijenu*** (engl. *Cost Function).* Postoje različite formule koje se koriste za definiciju ove cijene, a jedan od postupaka kojim se određuju težine w_i i prag b je algoritma ***inkrementalni spust gradijenta*** (engl. *Incremental Gradient Descent*) opisan u poglavlju 6.3.8.

V8.3 Duboko učenje

Duboko učenje (engl. *Deep Learning*), koje je danas posebno popularno, temelji se na višeslojnoj neuronskoj mreži koja između ulaznog i izlaznog sloja ima više skrivenih slojeva nego što je to uobičajeno kod neuralnih mreža. Budući da se često treniraju na velikom broju ulaznih podataka, ponekad je potrebno imati jako brzo računalo da bi se istrenirale. Čak i na brzim računalima ovaj proces može trajati danima.

V8.4 Implementacije neuronskih mreža

Programska biblioteka neuronskih mreža može se pronaći u raznim Python bibliotekama, npr.:

- ***OpenCV*** (engl. *Open Source Computer Library*) biblioteka za Python, C, C++, C# i Javu
- ***SciKit***-Learn biblioteka za Python
- ***WEKA*** - aplikacija za strojno učenje i sl.

Programske biblioteke dubokog učenja mogu se također pronaći u brojnim razvojnim okruženjima (engl. *Framework)*, npr.:

- ***Caffe framework***
- ***TensorFlow*** sustav za strojno učenje
- ***Theano*** biblioteka za Python i sl.

V8.5 Odabir ulaznih podataka

Neuronske mreže trebaju podatke za fazu treniranja. Ti su podaci obično dani kao vektori značajki uz koje je vezana oznaka klase.

Primjer ulaznih podataka dan je u tablici V8-1. U ovome primjeru redci u tablici (svi osim prvoga) predstavljaju piksele slike koji su određene kombinacije crvene, zelene i plave boje (engl. *RGB - Red Green Blue)* u 8-bitnom kodiranju koja pripada klasi na slici. Na primjer, prvi piksel čije su vrijednosti prikazane u tablici ima koordinate boja (85, 212, 55) i zna se da pripada klasi „vegetacija”. Što mislite koju bi od ponuđenih značajki bilo najbolje koristiti za klasifikaciju?

Tablica V8-1. Primjer ulaznih podataka za algoritme strojnog učenja

R	G	B	Klasa
85	212	55	Vegetacija
31	159	32	Vegetacija
31	153	194	Nebo
31	199	194	Nebo

Odabir najboljih značajki za klasifikaciju je područje istraživanja grane strojnog učenja koja se naziva ***odabir značajki*** (engl. *Feature Selection)*. Postoje različiti algoritmi koji pronalaze najbolje značajke, no najpoznatiji je ***PCA*** (engl. *Principal Component Analysis*) algoritam koji smo opisali u poglavlju 6.4.2.

V8.6 Zadaci

Zadatak 8.6.1. Napišite program u Pythonu koji se ponaša kao jednostavni perceptron. Proizvoljno odaberite temu o kojoj će vaš perceptron odlučivati.

Primjer koda:

```
print('Hoce li vrijeme biti suncano?')
x1 = input() ; čeka input sa tipkovnice
if (x1 == 'da'):
w1 = 2
else:
w1 = 0
```

Zadatak 8.6.2. Napišite program u Pythonu koji bi se ponašao kao sigmoidni neuron. Radi jednostavnosti, ograničite ulazne podatke na 0, 0,5 ili 1. Možete iskoristiti programski kod iz prethodnog zadataka, ali ga promijenite na način da program računa izlaz pomoću neke od aktivacijskih krivulja.

VJEŽBA 9 - UMJETNA INTELIGENCIJA U RAČUNALNIM IGRAMA

Umjetna inteligencija (UI) dosta se koristi prilikom izrade računalnih igara. U računalnim igrama prvenstveno se koristi tzv. slaba umjetna inteligencija koja je usmjerena na rješavanja problema

iz neke uske domene, a najčešće se koristi za vođenje ***automatskih virtualnih likova*** (engl. *NPC – Non-Player Characters*).

U računalnim igrama koriste se dva pristupa umjetnoj inteligenciji: deterministička i nedeterministička. U prošlosti se je češće koristila deterministička, zato što su stručnjaci za razvoj igara prvenstveno bili usredotočeni na što bolju grafiku u računalnoj igri, dok se danas obje vrste podjednako koriste.

V9.1 Deterministički postupci umjetne inteligencije

Karakteristike determinističkih postupaka umjetne inteligencije su:

- brza, jednostavna, lagana za implementaciju i testiranje
- programer mora programirati sve akcije i reakcije automatskih igrača
- igra može postati predvidljiva jer možete naučiti što očekivati od automatskog igrača budući da automatski igrač ne uči, tj. ne razvija se tijekom igre i
- igra može postati dosadna.

Primjer determinističke UI bio bi programiranje automatskog igrača koji prati računalni lik kojega korisnik kontrolira na način da uspoređuje koordinate korisničkog igrača s onima automatskog igrača – kada se preklope, automatski igrač je ulovio korisničkog igrača.

V9.2 Nedeterministički postupci umjetne inteligencije

Karakteristike nedeterminističkih postupaka umjetne inteligencije su:

- igra je nepredvidljiva
- automatski igrači uče od živih igrača, pa se razvijaju tijekom vremena
- igra ne postaje dosadna i
- igru nije jednostavno testirati.

V9.3 Tehnike umjetne inteligencije u računalnim igrama

U računalnim igrama koriste se različite tehnike umjetne inteligencije. Neke od njih smo već upoznali, a neke su specifične baš za računalne igre:

- pronalazak optimalnog puta različitim postupcima pretraživanja (engl. *Pathfinding)*
- logički automati koji se oslanjaju na neizrazitu logiku (engl. *Fuzzy State Machines)*
- konačni automati (engl. *Finite State Machines*)
- stabla ponašanja (engl. *Behavior Trees*)
- varanje (engl. *Cheating*) itd.

V9.3.1 Pronalazak optimalnog puta

Postupci pretraživanja koji se najčešće koriste za automatski pronalazak optimalnog puta u računalnim igrama su[122]:

[122] Vizualizacije većine ovih algoritama možete pronaći na sljedećem linku: https://www.redblobgames.com/pathfinding/a-star/introduction.html .

- ***slijepa pretraživanja***, npr. pretraživanje u širinu *(*engl. *Breadth First Search)* ili pretraživanje u dubinu *(*engl. *Depth First Search)*
- ***optimalno usmjereno pretraživanje***, npr. Dijkstra algoritam ili pretraživanje jednolikim troškom *(*engl. *Uniform Cost Search)*
- ***heurističko potraživanje***, npr. pohlepno pretraživanje *(*engl. *Greedy Search)*
- ***kombinacija optimalnog i heurističkog pretraživanja***, npr. A* pretraživanje.

Od svih nabrojenih postupaka pretraživanja koji se koriste za pronalazak optimalnog puta izdvojit ćemo možda najboljeg od njih, odnosno A* algoritam pretraživanja koji smo detaljno opisali u poglavlju 3.6.3. Ovaj algoritam traži put od početnog bloka do zadanog cilja na način da pregledava susjedne blokove i pomakne se na onaj koji ima najmanju cijenu. Cijena je u ovome slučaju definirana kao:

$$f(s) = c + h$$

gdje je:

- *c* – cijena koju je potrebno stvarno platiti da bi se od trenutnog bloka prešlo na susjedni (ne mora biti jedinstvena za sve blokove) i
- *h* – heuristička procjena puta (broja blokova) koje je potrebno prijeći od susjednog bloka do cilja.

Heuristička procjena blokova koje je potrebno prijeći od određenog bloka do cilja može se izračunati na više načina (detalji u poglavlju 3.5), a jedna od najčešće korištenih je Manhattanska udaljenost:

$$h = |\ x - x_{cilj}\ | + |\ y - y_{cilj}\ |$$

Primjer pronalaska optimalnog puta pomoću A* algoritma dan je na slici V9-1. Na slici se pretpostavlja da su na početku sva polja slobodna osim onih označenih sa #, a cijena prelaska iz jedno polje u drugo je ista za sva polja.

	0	1	2	3	4
0	#				**Cilj** g = 10 h = |0-0|+|4-4| = 0 **f = 10**
1	#			g = 10 h = |1-0|+|3-4| = 2 **f = 11**	g = 9 h = |1-0|+|4-4| = 1 **f = 10**
2	#	#	#	#	g = 8 h = |2-0|+|4-4| = 2 **f = 10**
3			#	g = 8 h = |3-0|+|3-4| = 4 **f = 12**	g = 7 h = |3-0|+|4-4| = 3 **f = 10**
4	g = 2 h = |4-0|+|0-4| = 8 **f = 10**	g = 3 h = |4-0|+|1-4| = 7 **f = 10**	g = 4 h = |4-0|+|2-4| = 6 **f = 10**	g = 5 h = |4-0|+|3-4| = 5 **f = 10**	g = 6 h = |4-0|+|4-4| = 4 **f = 10**
5	g = 1 h = |5-0|+|0-4| = 9 **f = 10**	g = 2 h = |5-0|+|1-4| = 8 **f = 10**	g = 3 h = |5-0|+|2-4| = 7 **f = 10**	g = 4 h = |5-0|+|3-4| = 6 **f = 10**	g = 5 h = |5-0|+|4-4| = 5 **f = 10**
6	**Početni blok**	#	g = 4 h = |6-0|+|2-4| = 8 **f = 12**	#	#

Slika V9-1. Primjer pronalaska optimalnog puta pomoću A algoritma*

V9.3.2 Konačni automati

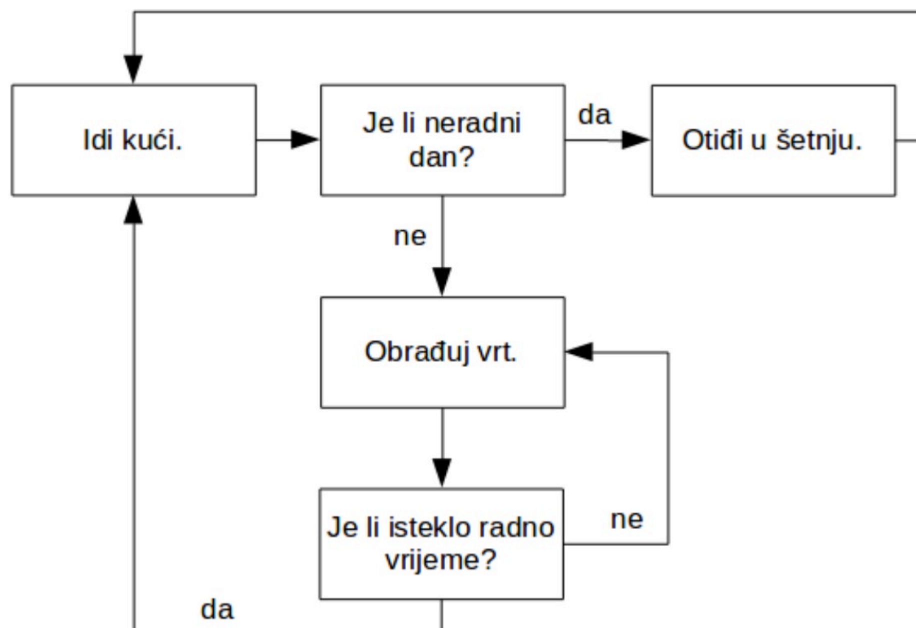
Kod upotrebe konačnih automata u računalnim igrama određuju se sve situacije u kojima se računalni lik može zateći, te se programiraju reakcije tog lika u svakoj od tih situacija. Ovo znači da su sve reakcije automatskih igrača unaprijed zadane. Zbog toga oni spadaju u determinističke postupke umjetne inteligencije, pa su nedostaci:

- ***predvidljivost*** – nakon nekog vremena igrač može pogoditi što će automatski lik učiniti u određenoj situaciji i
- ***složenost*** – problemi kod implementacije prilikom dodavanja novih stanja.

V9.3.3 Stabla ponašanja

Stabla ponašanja (engl. *Behavior Trees*) su potkategorija konačnih automata. Blokovi od kojih su ovi automati izgrađeni su akcije, a ne stanja. Jednostavniji su za dodavanje novih blokova od konačnih automata. Često se koriste zajedno s konačnim automatima.

Primjer stabla ponašanja za jednostavnu računalnu igricu dan je na slici V9-2.



Slika V9-2. Stablo ponašanja za jednostavnu računalnu igru

V9.3.4 Logički automati koji se oslanjaju na neizrazitu logiku

Logički automati koji se oslanjaju na ***neizrazitu logiku*** (engl. *Fuzzy Logic)* dopuštaju više mogućih stanja. Mogu koristiti vjerojatnosti da bi odlučili u koje će stanje prijeći, ili mogu slučajno odlučivati u koje će od mogućih stanja prijeći. Igre koje koriste ovakvu logiku su nepredvidljive. Dobro modeliraju ljude, jer su i oni često nepredvidljivi. Grupe likova u igrama mogu se dobro modelirati pomoću ovih automata jer će se likovi ponašati različito (a ne morate ih sve posebno programirati). Zbog toga oni spadaju u nedeterminističke postupke umjetne inteligencije.

V9.3.5 Varanje

Varanje se u računalnim igrama događa kada automatski igrač ima pristup informacijama vezanim za igrača kojeg korisnik kontrolira, te ih koristi kako bi ga pobijedio. Kada korisnik shvati da računalo vara, može odustati od igre jer su mu šanse za pobjedu male ili nikakve. Zbog ovoga je varanje potrebno držati na određenoj razini koja bi igru učinila zanimljivom i napetom.

V9.4 PYGAME BIBLIOTEKA

Pygame je biblioteka za Python koja implementira različite funkcije za stvaranje videoigara. Izvrstan *tutorial* za izradu videoigara u Pythonu uz korištenje Pygame biblioteke dostupan je na linku:

https://www.youtube.com/channel/UCJbPGzawDH1njbqV-D5HqKw.

V9.5 Zadaci

Zadatak 9.5.1. Pomoću A* algoritma u Pythonu pronađite put kroz labirint prikazan u tablici V8-1. Pretpostavite da u tablici 0 predstavlja prazno polje, 1 početno polje, 2 cilj, 3 prepreku, te 4 polje koje ste već posjetili. Prilikom pronalaska optimalnog puta kroz labirint vaš algoritam mora slijediti određena pravila: dozvoljeno je kretati se samo lijevo, desno, gore i dolje, te nije dozvoljeno 2 puta posjetiti isto polje. U koliko koraka algoritam dođe do cilja?

Tablica V9-1 Labirint vezan uz zadatak 1

0	0	3	0	2
0	0	0	0	0
0	3	3	0	0
1	0	0	0	0

Stvaranje matrice u Pythonu:

labirint = numpy.matrix([[0, 0, 3, 0, 2], [0, 0, 0, 0, 0], [0, 3, 3, 0, 0], [1, 0, 0, 0, 0]])

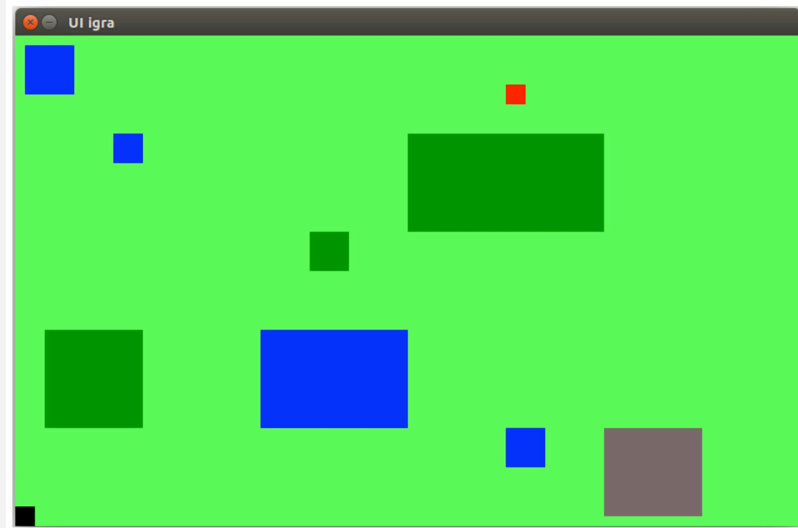
Zadatak 9.5.2. Nadogradite algoritam iz zadatka 1 na način da je dozvoljeno i dijagonalno kretanje. Promijenite funkciju za računanje heuristike – nemojte koristiti Manhattan udaljenost, nego uzmite u obzir i dijagonalno kretanje pa koristite Euklidsku udaljenost. U koliko koraka algoritam dođe do cilja?

Zadatak 9.5.3. Nadogradite algoritam iz zadatka 2 na način da je moguće posjećivati već posjećena polja, te ga testirajte na labirintu prikazanom u tablici V9-2. U koliko koraka algoritam dođe do cilja?

Tablica V8-2 Labirint vezan uz zadatak 3

0	0	3	0	3
0	0	0	0	0
0	0	3	0	0
1	3	0	0	0

Zadatak 9.5.4. U Pythonu napravite jednostavnu igru u kojoj računalni lik (možete ga predstaviti pomoću bloka) sam pronalazi put kroz labirint pomoću A* algoritma. Računalni lik treba doći do predefiniranog cilja (kvadrat crvene boje). Sami definirajte statične kvadrate u labirintu koji će predstavljati prepreke. Na primjer, plavi kvadrati mogu predstavljati vodu, zeleni šumu, a sivi planine. Primjer prozora u kojemu bi bila ovakva igra dan je na slici V9-3.



Slika V9-3. Primjer jednostavne igre u Pythonu

Zadatak 9.5.5. Modificirajte algoritam iz prethodnog zadatka na način da je moguće kretanje preko prepreka, ali cijena kretanja preko prepreka nije 1, nego veća od 1. Na primjer, cijena kretanja kroz šumu je 2, preko vode 3, a preko planine 4. Ove cijene definirajte na početku programa.

Zadatak 9.5.6. Modificirajte algoritam iz prethodnog zadatka na način da računalni lik i prepreke ne predstavljaju blokovi, već slike. Slike možete dobiti na dva načina: u GIMP-u sami nacrtajte računalni lik, šumu, planinu i vodu, ili na internetu pronađite slike koje su vam potrebne, a koje imaju edukacijsku licencu.

VJEŽBA 10 - OBRADA PRIRODNOG JEZIKA

Obrada prirodnog jezika (engl. *NLP – Natural Language Processing)* je dio umjetne inteligencije koji se bavi analizom i razumijevanjem napisanog ili izgovorenog teksta. U NLP postupke spada:

- automatsko sažimanje teksta
- potraga za sličnim dokumentima
- prepoznavanje govora
- automatsko generiranje teksta, itd.

U ovoj laboratorijskoj vježbi napravit ćemo jednostavni program u Pythonu koji bi iz pisanih recenzija nekog filma mogao donijeti odluku o tome je li film bio uspješan ili ne.

V10.1 Zadaci

Zadatak 10.1.1. Kreirajte recenzije.txt datoteku koja sadrži par različitih recenzija za neki film. Svaki redak te datoteke neka sadrži jednu recenziju. Možete koristiti recenzije dane u nastavku teksta ili napisati sami svoje recenzije.

Primjeri recenzija nekog filma:

Best movie ever! I'm really glad I watched it. It was such fun!

It was ok. Nothing special. I've seen worse movies.

Horrible movie. Hated every second of it. Would not recommend to anyone.

I had a blast while eatching this movie. It was so funny. My family loved it. It is a great family movie.

Cool movie. Loved the actors and the plot.

Super! Loved it!

Could not wait for this movie to end - it was so boring.

Loved the main actress! She really portrayed the character well.

I recommended this movie to all my friends and family and they all loved it.

It was so funny that I wish they would make a sequel. Would definitely recommend.

Primjer učitavanja tekstualne datoteke u Pythonu:

```
datoteka = "recenzije.txt"
# ucitavanje tekstualne datoteke
with open(datoteka) as f:
sadrzaj = f.read().splitlines()
print(sadrzaj)
print("Broj recenzija u datoteci: " + str(len(sadrzaj)))
for i in range (0, len(sadrzaj)):
print("Recenzija: " + str(sadrzaj[i]))
sadrzaj_bez_delimitera = sadrzaj[i].replace('.', '')
sadrzaj_bez_delimitera = sadrzaj_bez_delimitera.replace(',', '')
sadrzaj_bez_delimitera = sadrzaj_bez_delimitera.replace('!', '')
sadrzaj_bez_delimitera = sadrzaj_bez_delimitera.replace('-', '')
print("Recenzija bez delimitera: " + str(sadrzaj_bez_delimitera))
rijeci_iz_trenutne_recenzije = sadrzaj_bez_delimitera.split()
print("Rijeci u recenziji: " + str(rijeci_iz_trenutne_recenzije))
```

Zadatak 10.1.2. Napišite program u Pythonu koji će učitati recenzije iz datoteke *recenzije.txt* i ispisati ih.

Zadatak 10.1.3. Proučite recenzije u datoteci *recenzije.txt*. Koje riječi označavaju pozitivno mišljenje o filmu, a koje negativno? Spremite te riječi u dvije zasebne liste ili pak spremite svaku riječ u zasebni string. Ove riječi sada predstavljaju vaš ***rječnik*** ili vokabular (engl. *Vocabulary).*

Zadatak 10.1.4. Napišite petlju u Pythonu koja prolazi kroz svaku recenziju posebno i bilježi koliko puta se je u toj recenziji pojavila neka pozitivna riječ, te koliko puta se je pojavila neka negativna riječ. Ako su se više puta u recenziji pojavile pozitivne riječi, ocijenite tu recenziju kao pozitivnu. Ako su se više puta u recenziji pojavile negativne riječi, ocijenite tu recenziju kao negativnu. Ako se u recenziji nije pojavila nijedna riječ iz vašeg rječnika, ignorirajte tu recenziju.

Zadatak 10.1.5. Odredite postotak korisnika koji su filmu dali pozitivnu recenziju, te postotak korisnika koji su filmu dali negativnu recenziju.

....

CIP – Katalogizacija u publikaciji
S V E U Č I L I Š N A   K N J I Ž N I C A   U   S P L I T U

UDK 004.8(075.8)

STIPANIČEV, Darko
Uvod u umjetnu inteligenciju / Darko Stipaničev, Ljiljana Šerić, Maja Braović -
Split: Fakultet elektrotehnike, strojarstva i brodogradnje, 2021. -
(Udžbenici Sveučilišta u Splitu = Manualia universitatis studiorum Spalatiensis)

Bibliografija.

ISBN 978-953-290-108-5

1. Šerić, Ljiljana 2. Braović, Maja
I. Umjetna inteligencija – Udžbenici

190215059
....
ISBN 978-953-290-108-5